Specification, Verification and Optimisation of Business Processes

A Unified Framework

Herbert, Luke Thomas

Publication date:
2014

Document Version
Publisher's PDF, also known as Version of record

Specification, Verification and Optimisation of Business Processes

A Unified Framework

Luke Herbert

Kongens Lyngby 2014
COMPUTE-PhD-2014-303

COMPUTE-PhD-2014-303

ISSN: 0909-3192
Technical University of Denmark
DTU COMPUTE - Department of Applied Mathematics and Computer Science
Richard Petersens Plads
Building 324
DK-2800 Kongens Lyngby
Denmark

Phone +(45) 45 25 33 51
Phone +(45) 45 88 26 73

compute@compute.dtu.dk
www.compute.dtu.dk

Preface

This thesis was prepared under the guidance of Professor Robin Sharp at the department of Informatics at the Technical University of Denmark in fulfilment of the requirements for acquiring a Ph.D. degree in Informatics.

The work presented in this thesis was jointly funded by Intelligent Hospital Systems and the Technical University of Denmark.

This second edition incorporates corrections suggested by the PhD examiners: A denotational semantics of Core BPMN process models has been added, and a number of grammatical and typographical errors have been corrected.

Lyngby, 20-March-2014
Second Edition: March 2015

Luke Herbert

Table of Contents

List of Figures

List of Acronyms

The following arconyms are used in this thesis:

A | B | C | D | E | F | G | I | L | M | O | P | S | T | U | W | X | Y

A
API Application Programming Interface
ARIS Architecture of Integrated Information Systems

B
BDD Binary Decision Diagram
BPD Business Process Diagram
BPEL Business Process Execution Language
BPM Business Process Management
BPMN Business Process Model and Notation
BPR Business Process Re-Engineering

C
CCS Calculus of Communicating Systems
CEGAR Counter Example Guided Abstraction Refinement
COWS Calculus for Orchestration of Web Services
CSL Continuous Stochastic Logic
CSP Communicating Sequential Processes
CTL Computation Tree Logic
CTMC Continuous Time Markov Chain

D
DC Duration Calculus
DTMC Discrete-time Markov Chain
DTU The Technical University of Denmark

E
EPC Event-driven Process Chain

F
FDR Failures-Divergences Refinement
FTA Fault Tree Analysis
FTA Fault Tree Analysis

G
GNU-GPL GNU General Public License

I
IMC Interactive Markov Chains
IT Information Technology
IV Intravenous bag

L
LTL Linear Temporal Logic
LTS Labelled Transition System

M
MDP Markov Decision Process
MODEST The Modelling and Description Language for Stochastic and Timed Systems
MOTOR MOdest TOol EnviRonment
MRMC Markov Reward Model Checker
MTBDD Multiple Terminal Binary Decision Diagram

O
OBDD Ordered Binary Decision Diagram
OMG Object Management Group

P
PCTL Probabilistic Computation Tree Logic
PEPA Performance Evaluation Process Algebra
PEPS Performance Evaluation of Parallel Programs
PRISM Probabilistic Symbolic Model Checker
PRLC Business Process Re-engineering Life Cycle
PTA Probabilistic Timed Automata

S
SBOAT Stochastic BPMN Optimisation Analysis Tool
SHA Stochastic Hybrid Automata

T
TPTP Thousands of Problems for Theorem Provers

U

UML Unified Modelling Language
UML-MARTE Modelling and Analysis of Real Time and Embedded systems
UML-SysML Unified Modeling Language Systems Modeling Language

W
WMTL Weighted Metric Temporal Logic
WSBPEL The Web Services Business Process Execution Language

X
XML eXtensible Markup Language
XPDL XML Process Definition Language

Y
YAWL Yet Another Workflow Language

Summary (English)

This thesis develops a unified framework wherein to specify, verify and optimise stochastic business processes.

This framework provides for the modelling of business processes via a mathematical structure which captures business processes as a series of connected activities. This structure is extended with stochastic branching, message passing and reward annotations which allow for the modelling of resources consumed during the execution of a business process. Further, it is shown how this structure can be used to formalise the established business process modelling language Business Process Model and Notation (BPMN).

The automated analysis of business processes is done by means of quantitative probabilistic model checking which allows verification of validation and performance properties through use of an algorithm for the translation of business process models into a format amenable to model checking. This allows for a rich set of both qualitative and quantitative properties of a business process to be precisely determined in an automated fashion directly from the model of the business process.

A number of advanced applications of this framework are presented which allow for automated fault tree analysis and the automated optimisation of business processes by means of an evolutionary algorithm.

This work is motivated by problems that stem from the healthcare sector, and examples encountered in this field are used to illustrate these developments.

Summary (Danish)

Denne afhandling udvikler et forenet framework til at specificere, kontrollere og optimere stokastiske forretningsprocesser.

Dette framework giver mulighed for modellering af forretningsprocesser via en matematisk struktur, der indfanger forretningsprocesser som en række forbundne aktiviteter. Denne struktur er udvidet med stokastisk forgrening, besked overførelse og belønningsannotationer, der giver mulighed for modellering af ressourcerne forbrugt i løbet af udførelsen af en forretningsproces. Endvidere er det vist, hvordan denne struktur kan bruges til at formalisere det etablerede forretningsmodelleringssprog Business Process Model and Notation (BPMN).

Den automatiserede analyse af forretningsprocesser sker ved hjælp af kvantitativ probabilistisk model checking, som åbner mulighed for kontrol af validerings-og styrkemæssige egenskaber ved brug af en algoritme til oversættelse af forretningsprocessmodeller til et format hvor model checking teknikker kan anvendes. Dette giver mulighed for at et rigt sæt af både kvalitative og kvantitative egenskaber for en forretningsproces kan bestemmes præcist i en automatiseret måde direkte fra modellen af forretningsprocessen.

En række avancerede anvendelser af dette framework præsenteres som giver mulighed for automatiseret fault tree analyse og automatiseret optimering af forretningsprocesser ved hjælp af en evolutionær algoritme.

Dette arbejde er motiveret af problemer, der stammer fra sundhedssektoren, og eksempler fra dette område anvendes til at illustrere denne udvikling.

Acknowledgements

Firstly, I would like to thank Intelligent Hospital Systems, specifically CEO Dr. Niels-Erik Hansen and CTO Thom Doherty, for their extensive advice on their RIVA product and the complexities of introducing automation in a hospital environment and the market demands this produces. It was Dr. Hansen's decision that Intelligent Hospital Systems could benefit from this work that gave me the opportunity to do this project, and for that I will be forever grateful. I would like to thank not only Dr. Hansen but the entire Hansen family for their support during this project.

Furthermore, I would like to thank Robin Sharp for the countless invaluable discussions we have had during this project. These lively meetings have been immensely stimulating and our discussions have stretched over a broad spectrum of scientific questions and have given me a lifetime of interesting problems to ponder. The friendship he has shown me and the unending supply of good humour have been essential for completing this project. Working with him has been a great privilege and I hope to make him proud in my future academic endeavours. Robin has given me a chance to start a new life and he has set me on a new path where building a family and finding real happiness are within my grasp. I look forward to many more discussions, and will one day show him that something original can be proven with category theory.

I would like to thank my father for bringing home a Commodore VIC-20 all those years ago and first introducing me to a world where I could learn at my own pace and in which only my imagination, and the bounds of computational complexity, would set limits. Though our relationship is complex I wish him well and I think he knows I am there for him as the situation allows. I would also like to thank my grandmother Rosemary, for her support during my PhD and the polite interest she has always shown when I go on for too long about my work.

I would like to thank my mother for always encouraging me to study, learn and develop and for the many sacrifices she made to ensure I received a good education. She has encouraged me in everything that I have done, despite her own beliefs and the many times I have stumbled. She has been and remains a wonderful mother and I owe her so much.

I would like to thank Crystal for her great graphic design work, including making the SBOAT logo and icons, and for her tireless efforts in helping me with my endless spelling and grammar mistakes. She, and her family, have always been a joy to visit and made me feel very welcome on every occasion.

I would like to thank my friends for not knowing anything about what I study and creating an environment where I could be free of informatics and thoughts of my PhD project. I would like to thank them for their faith in my abilities and the certainty that whatever happens in my life I will always be able to depend on them.

Furthermore, I would like to thank Tina Andersen for helping me during my formative years. You taught me many things, but your faith in me and belief that I should follow my dreams helped put me on the path I am today. You taught me some of the hardest lessons I have learned, but also to be playful, creative and to always be true to myself. I will always cherish our time together and wish you the very best in life.

Finally, I would like to thank Zaza for convincing me that I could do a PhD and her support throughout the entire process. She has helped me every step of this long journey, encouraged me when I thought doing this was hopeless, shared the successes I have had, and often caught me smiling for *"no reason"*, never realising that I was thinking about her. Her faith in me makes me want to be the man she believes I can be and I will do anything to help fulfil her dreams. She has listened so patiently when I have talked about my work (and I am sure she never want to hear me speak about business processes every again), inspired me countless times by asking brilliant questions, and shown me love like I have never known before.

Thank you, all of you. Completing this project is only the beginning of what comes next.

Chapter 1

Introduction

"This is no defeatism...The declaration of our own nature and the attempt to build up an enclave of organization in the face of nature's overwhelming tendency to disorder is an insolence against the gods and the iron necessity that they impose. Here lies tragedy, but here lies glory too..." (Norbert Wiener 1948)

Overview

This chapter states the objectives for the thesis, introduces the motivating context of robot assisted workflows in hospital environments, and provides an overview of the structure of this thesis.

1.1 Motivation

The original motivation for this project stems from a problem in the healthcare sector.

A significant proportion of the treatment performed in modern hospitals relies on the adaptive application of various intravenous medicines, which are mostly in liquid form and prepared in central pharmacies serving a number of departments. Strict safety requirements mean that the process of preparation is labour intensive. Nevertheless, with human actors they still remain error-prone [52], [128].

The management of hospital pharmacy operations is often not able to efficiently deliver medications. Current product preparation workflows rely on manual processing, which is often compromised by poor inventory management [79], [128]. Pharmacies tend to employ production techniques abandoned by other industries; including duplicating inventory, inflexible batch processing, and over production where doses are often prepared in anticipation of a need that may not be realised. These processes contribute to waste, as sometimes doses go missing, or there are attempts to improve the delivery times by having redundant distributed inventories to avoid transport issues [136]. Standard doses are used, rather than patient specific supplies which level work load and solve distribution delay concerns [128].

This situation has become increasingly untenable, and a solution has been sought by the introduction of automation into the medication management and dispensing processes. Initial deployments have delivered significant improvements in specific pharmacy tasks by improving the quality of care through dramatically increased safety for both staff and patients [67], [212]. New pharmacy capabilities such as customised medicine have also become feasible [67].

Robotic intravenous drug preparation systems are used by hospital pharmacies to automatically and accurately prepare IV syringes and bags. An example of such a system is shown in Figure 1.1. This system consists of a robotic arm inside a clean-room environment which moves drugs, syringes and IV bags between

separate stations. The stations perform steps in the preparation of drugs such as filling syringes, mixing/diluting drugs, and heating drugs. The system takes in standard syringes, IV bags, and vials of drugs and produces ready-to-use labelled doses of drugs. The automation of the repetitive and complex tasks involved in drug preparation reduces the incidence of errors and contamination [67], [128].

A fundamental prerequisite is that the system meets regulatory requirements for safety, while improving the efficiency and effectiveness of the pharmacy within a hospital. This is very important when preparing chemotherapeutic drugs as the system provides significantly increased safety for the technician, who could be endangered when preparing these drugs. Such systems also need to operate within a strict performance profile because the chemotherapeutic versions of these drugs can have very short half-lives.

This technology, however, has been disruptive in the traditional pharmacy environment and customers have not been aware of how best to integrate this product into their existing business processes. A system such as that illustrated in Figure 1.1 consists of many interacting, distributed elements subject to real time constraints and with behaviours that may be stochastic in nature. The environment in which a medical robot operates is also stochastic and subject to quantitative resource constraints. In this environment the product is integrated into current business processes in an extremely conservative fashion. The approach is often to simply swap out an existing pharmacy with the new automated solution, which does not achieve the full gains that the technology potentially enables.

This thesis presents examples, drawn from the experiences of a commercial vendor trying to solve the problem of safely realising the full efficiency gains that are expected by integrating robotics into a hospital environment. The main reason for employing this environment to demonstrate the ideas in this thesis, is that business process modelling has seen wide adoption in this sector [237], [249]. To place the examples and developments presented in this thesis in context, the specific problems faced by this partner are briefly described below.

A simple example of the type of problems faced by the commercial vendor producing automated intravenous drug preparation systems, was that current standard hospital practice involves producing a large batch of drugs in the morning, which are then stored and administered during the day according to the patient's treatment plan. Should a patient's condition change, any drugs prepared for that patient are destroyed, leading to substantial waste [67]. However, the automated system's fast and safe processing of drugs enables the adoption of *just-in-time production* [263], where drugs are prepared on an as-

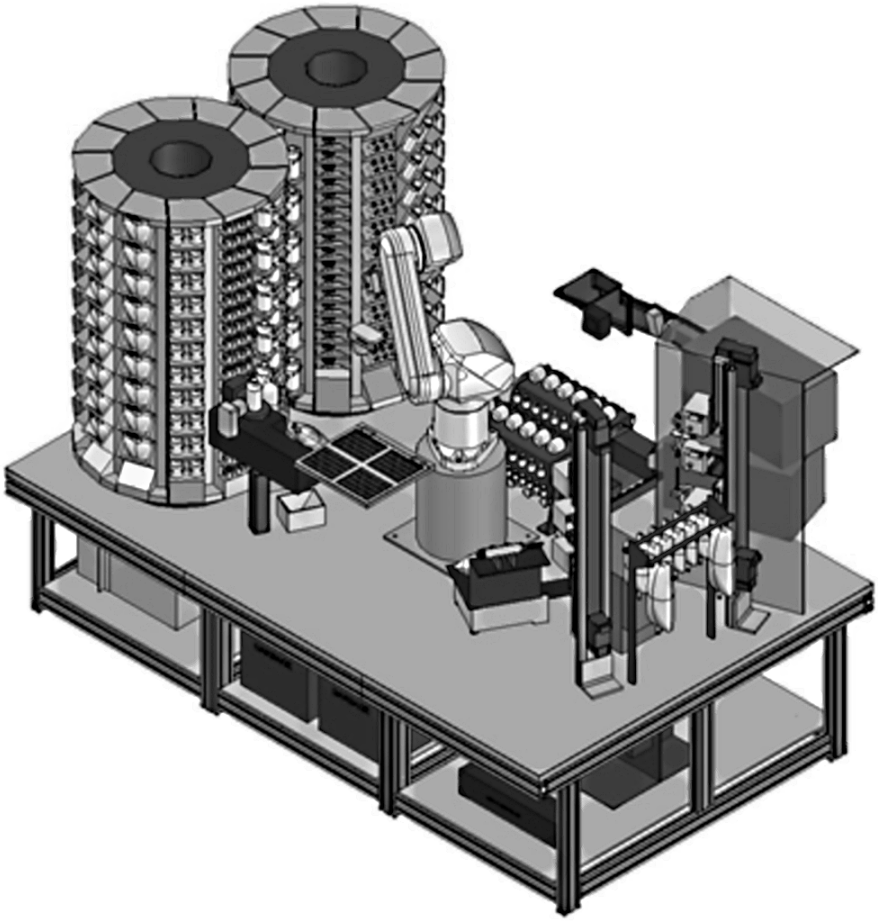


Figure 1.1: *Interior of an automated fluid drug preparation system. The central robot arm moves materials between various processing stations.*

needed basis just prior to use. Discovering that the system enables this change in a business process was non-obvious, and motivates the search for an automated way to discover configurations which fully realise potential efficiency gains.

Ensuring that these systems make efficient use of resources and perform in such a way so as not to violate any established safety standards or regulatory guidelines is paramount, as situations may well be encountered where faults can lead to the loss of human life. Faults may be complex in nature and systems may not halt their operations when a fault has occurred, leading to further complexity in the analysis of such systems. The earlier it is possible to anticipate and

counterbalance the risks present in a new business process then the greater the benefit of a formal modelling approach. Design time support for the analysis of hospital operations, and how they would be affected by the introduction of new systems, is essential to make efficient use of these systems. However, such an analysis must be able to validate safety properties, performance properties, and provide support to adjust current operations to find the optimal way to employ the new system.

In the healthcare sector in many jurisdictions a fault tree analysis [97] is frequently a regulatory requirement when introducing a new device into a medical workflow. Hence being able to automatically generate fault trees at an early stage in the business processes development can help identify possible complex faults before a process is employed, thus leading to increased safety and reducing time and cost needed to construct dependable business processes. Further, automated optimisation of business processes early on in their development can help keep individual processes simple and ensure that deployed processes are more efficient. However, it should be stressed that these needs not only exist at design time but that fault tree analysis and optimisation can be useful throughout the development of a business process.

Unfortunately the specific examples originally used to develop the ideas presented in this thesis can not be included due to commercial concerns of the partner company *Intelligent Hospital Systems*. However, the examples given are representative of the models encountered in developing this framework. In Section 9.3 of Chapter 9 the scalability of the techniques developed is examined for a range of randomly generated models.

1.2 Objectives of the Thesis

The main objective of this thesis is to determine:

How to specify, verify and optimise stochastic business processes within a unified design time framework.

Based on the motivating context of Section 1.1, there is a need for a rich modelling formalism which can allow for describing processes with a combination of stochastic and non-deterministic behaviour, and which can account for resources consumed during execution of the process. These requirements motivate Objective 1.

Central to the work of this thesis is enabling the application of the powerful technique of quantitative probabilistic model checking to the analysis of business processes of the type described in Section 1.1. Useful analysis of such processes most be able to account for both functional and non-functional requirements of a process and as such this motivates Objective 2. Finally, it is a requirement of the partner company to perform a number of specific analyses of new business processes, motivated by the need to comply with regulatory requirements and as part of building the case for efficiency gains. These requirements are expressed as Objective 3.

A more extensive statement of the objectives is as follows:

1. **Modelling:** To develop a formalism for the specification of business processes which is able to address their complexities and can be formally analysed. This formalism should:
   (a) Allow for the modelling of both non-determinism and probabilistic decision points.
   (b) Allow for modelling numerical data and resources associated with a business process.
   (c) By means of the imposition of structural semantics serve as a basis for the formalisation of an established business process modelling language which is familiar to business practitioners.
2. **Analysis:** To develop a method of analysis for systems of business processes which allows for automated formal mathematical verification of:
   (a) Functional safety properties which assert that the system always stays within some allowed region, such as defined by regulatory requirements, of business processes.
   (b) Non functional qualities of business processes such as timing properties or the determination of resources consumed by business processes.
   (c) Probability bounds on functional and non-functional properties for business processes which exhibit stochastic behaviour.
3. **Extended Analysis:** To employ the developed verification technique to allow for construction of advanced design time analysis techniques for business processes, which specifically allow for:
   (a) Automatic synthesis of execution schedules for business processes which are subject to combined functional and non-functional requirements.
   (b) The generation of industry standard fault trees which can assist in the analysis of the effects of combined faults within a business processes.
   (c) The optimisation of existing business processes.

This set of objectives is addressed through a framework for the analysis and optimisation of business processes using quantitative model checking methods. This framework can be viewed as a implementation of, and at times, an alternative approach to, the ideas of Business Process Management (BPM) (described in Section 2.2). This framework allows for the sophisticated analysis of business processes already in the design phase, including quantitative and stochastic properties. This in turn has the potential to enable optimisation of business processes by using quantitative data, from for example equipment specifications, industry standards or common practice without the need for data mining event logs which characterizes the traditional BPM approach, potentially allowing more safe and efficient processes to be deployed at the outset.

The extent to which the thesis meets these goals is evaluated in Section 10.2.

1.3 Organisation of the Thesis

The structure of this thesis reflects the approach taken to meet these objectives, and is illustrated in Figure 1.2. In this figure the lightest grey boxes represent foundational material only briefly covered here so as to establish context, but not explained in depth. The darker boxes indicate areas where the thesis contains more original work. The black boxes represent advanced applications of these ideas and are the culmination of the work in this thesis.

The chapters of this thesis contain the following material:

Chapter 1 Introduction:
This chapter states the objectives for the thesis and introduces the motivating context of robot assisted workflows in hospital environments and provides an overview of the structure of this thesis.

Chapter 2 Business Processes:
This chapter defines business processes and discusses their complexities. The approach of business process management is introduced to place the developments of this thesis in a management engineering theory context. An overview of the main approaches to modelling business processes is provided and a set of requirements for the modelling of business processes within the motivating context are given. An overview of established business process modelling languages is presented and the choice of the Business Process Model and Notation (BPMN) [207] language as a suitable

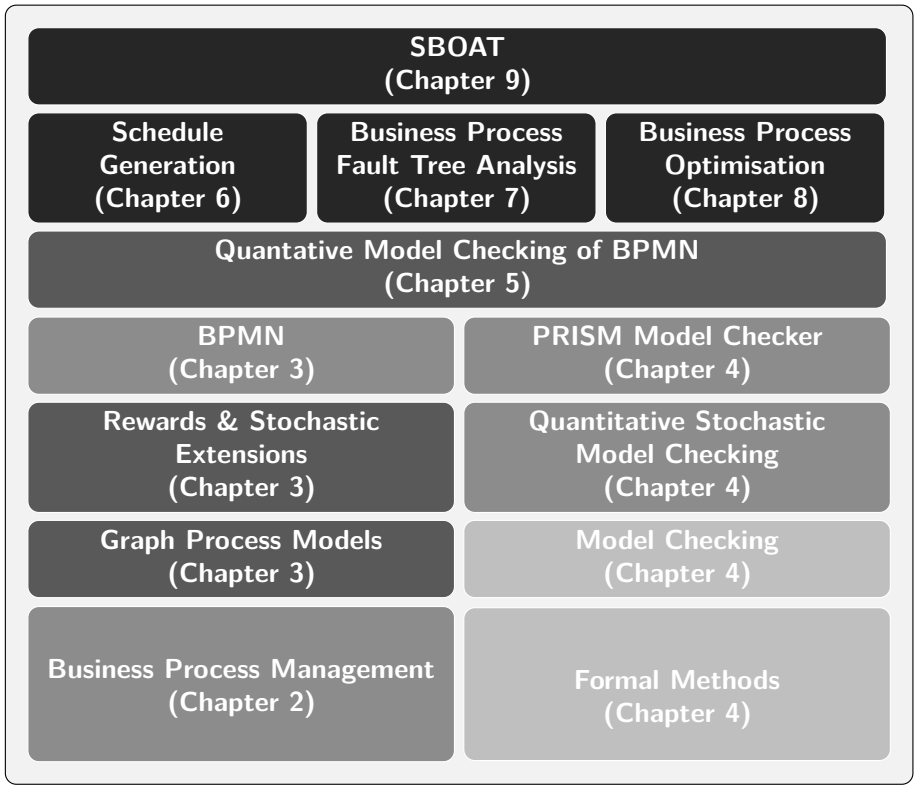


Figure 1.2: *Overview of the structure of the thesis.*

modelling language to employ as a basis for further development in this thesis.

Chapter 3 Modelling Business Processes:
This chapter defines the formalism of *process graphs* that allow for the modelling of business processes as a graph representing sequences of tasks. This structure is extended to allow for the modelling of probabilistic and non-deterministic branching, in line with Objective 1a. Objective 1b is met by defining quantitative data annotations for both nodes and vertices of a process graph. It is shown how these can be combined to approximate the accumulation of stochastic rewards given by an arbitrary continuous distribution. The synchronization of separate models is addressed by means of a message passing mechanism.

Process graphs are then employed as a basis for defining a notation and structural semantics for a central fragment of the BPMN [207] modelling language which meets Objective 1. The approach taken is to partition the nodes that form a process graph and then impose a set of structural semantic rules which exclude, from the perspective of modelling business processes, ill-formed graphs. Note that in this chapter the focus is on defining a notation for description of business processes with a minimum of semantic interpretation imposed upon the notation. However, a basis denotational semantics for process graphs in terms of execution traces is defined. The developments in this chapter are illustrated with an example drawn from the scenario described in Section 1.1.

Chapter 4 Formal Methods:
This chapter provides a brief overview of the main formal verification approaches for stochastic systems. These approaches are evaluated and the choice of model checking for analysis of business processes is justified. The dominant software tools for quantitative model checking are reviewed. Discussion of their key strengths and weaknesses motivate the choice of the Probabilistic Symbolic Model Checker (PRISM) model checker to perform analysis of business processes.

Chapter 5 Analysing Business Processes:
Objectives 2a-2c are addressed in this chapter. An algorithm is developed for the translation of process graph based BPMN models into PRISM models, and the algorithm's termination and complexity bounds are detailed. Pre- and post- processing steps that enhance the precision of the analysis and which allow for bounded rewards values are presented. In this chapter a semantic interpretation is imposed on the execution of BPMN models by the PRISM model checker, the ramifications of this interpretation and how it differs from what is defined in the BPMN execution semantics is discussed.

The application of this analysis approach is illustrated using the example introduced in Chapter 3, where a range of properties are verified, and provisioning of resources is demonstrated.

Chapter 6 Schedule Generation:
This chapter demonstrated how the analysis can be used to synthesise a schedule for the execution of a business process, subject to both functional and non-functional requirements, which addresses Objective 3a.

Chapter 7 Fault Tree Generation:
In this chapter, Objective 3b is addressed by further extending the process graph formalism introduced in Chapter 3 to include states that capture *fail-stop* and *fail-continue* behaviour. The injection of fault states is defined,

and an algorithm is presented which employs the methods of Chapter 5 for the automatic generation of industry standard fault trees from fault-annotated models. This chapter also shows how the value of reward values can be calculated as annotations at points of failure, and builds a fault tree which includes the mean values of properties of interest at fault states. A limited example of fault tree analysis is used to illustrate this work.

Chapter 8 Optimisation of Business Processes:
The chapter discusses the automated optimisation of business processes, which permits the improvement of existing processes or the effective integration of new technology into an existing process, as required by Objective 3c. This is approached by defining a structure for expressing the optimization goals for a business process as a set of weighted analyses of properties of interest in a business process. This structure is used to construct a fitness score for a given business process with regard to the optimisation goals. The same ideas are employed to express the core functional requirements for a business process which must not be violated by an optimised variant.

An algorithm for the optimisation of business processes is then presented. This algorithm allows for an evolution-like process to be applied to a business process to arrive at a configuration that is well-suited to achieving optimal values of the properties of interest. The algorithm takes an existing business process and a set of optimisation goals and functional requirements, and produces an improved business process in the same process graph formalism as the input business model. The iterative algorithm generates variants of an existing business process, and, at each iteration, determines if they meet the functional requirements. If they do, and a higher fitness score is achieved, then this generation is used as the basis for the next iteration. Finally, some of the core parameters of this process are discussed.

Chapter 9 SBOAT:
This chapter introduces a prototype software tool, called *SBOAT* for *S*tochastic *B*PMN *O*ptimisation and *A*nalysis *T*ool, which provides an implementation of the ideas presented in this thesis. The main elements are a GUI for the modelling of workflows in the developed fragment of BPMN and implementations of the verification and analysis techniques developed. Within SBOAT, properties of interest are specified using the temporal logic Probabilistic Computation Tree Logic (PCTL) and employ stochastic model checking, by means of the model checker PRISM, to compute their exact values as described in the previous chapters.
SBOAT is employed to explore the scalability of the approach to business process analysis and optimisation. The chapter will conclude with a discussion of the tool's practical applicability and need for further development.

Chapter 10 Conclusions:
This chapter concludes the thesis and provides a brief summary of the main contributions of this work. The main strengths of the developed approach are discussed, together with areas that need further refinement. During this discussion some possibilities for further development of this framework are identified. Some final remarks address the broader perspectives of the work presented.

Appendix A Quantitative Model Checking Theory:
This appendix provides as formal foundational description of quantitative model checking is given with a focus on PCTL checking of Markov Decision processes.

Chapter 2

Business Processes

"Being good in business is the most fascinating kind of art. Making money is art and working is art and good business is the best art."
*(*Andy Warhol 1975*)*

Overview

This chapter introduces concepts from the management engineering of business processes and discusses their complexities. The approach of business process management places the ideas of this thesis in a management engineering theory context. An overview of the goals of business process modelling is presented along with a discussion of the main approaches to modelling business processes. This discussion is employed to motivate the definition of a set of requirements for a suitable modelling language for modelling the type of workflows which are the focus of this thesis.

The challenges with business processes and business process modelling in this context were first presented in [10], and later developed in [11], and subsequently refined in [14]. A number of issues in production processes where explored in [1].

2.1 Business Processes

A wide range of frameworks for the description of the behaviour of organisations has been proposed. A significant majority of these fundamentally view organisations as *activity systems* [65], [297]. First described by Porter in 1985, the concept of *activities* and its use in understanding competitive advantage [229] has become the dominant approach to understanding organisational behaviour. Later, Porter noted that the competitiveness of a specific business strategy rests on the uniqueness of these activities, and on the deliberate choice of different sets of activities to deliver a unique mix of value propositions [228].

According to Porter, *positioning choices* determine the specific set of activities an organisation performs and how they relate to one another. Since discrete activities often influence one another, a system level approach is implicit in the activities view of organisational behaviour which emphasizes their interdependence and gives rise to the notion of *fit* among activities [227]. Three categories of non-mutually-exclusive fit have been described. First category fit is consistency between each activity and strategy. Consistency ensures that the competitive advantage arising from activities accumulates, and does not erode or cancel out [227]. Second category fit occurs when activities are mutually reinforcing. Here activities are complements, so that the marginal value of one activity increases as the other activity is increased [189]. Finally, third category fit goes beyond activity reinforcement to produce global optimization, a system-level type of fit, which optimizes the entire set of activities to eliminate redundancies and minimize waste [227].

The related concept of *workflow* has existed for a long time. The origins of this concept can be traced back at least as far as Smith's *Wealth of Nations* [266] which introduced, in 1776, the first ideas of the division of labour in manufacturing.

Originally, one person would make one item from beginning to end in a cottage industry situation. Later, when factories became the norm, employing many people who all made items from beginning to end proved time-consuming and inefficient. Gilbreth's paper [113] *Process Charts - first steps to finding the one best way to do work* is credited with addressing this and developing the first method for documenting process flow, where a workflow is a specific sequence of actions intended to achieve one or more objectives. These objectives are typically presented within frameworks which dictate the behaviour of organisations in the form of a *business model* [65], [297]. This motivates the choice of activities performed and how they are combined to execute a given strategy.

Although business models and workflows have been integral to trading and economic behaviour since pre-classical times, their formalisation has only relatively recently given rise to the concept of *business processes*. The definition of what constitutes a business process is a subject of debate, and over the past century a wide range of definitions has been proposed [176]. However, the consensus is that a business process is a series of related activities aimed at executing the components of a business model in a measurable manner. The notion of measurable quantities associated with the activities is the main distinction from a workflow, although the terms are frequently used interchangeably and a consistent terminology has never been established [65], [176].

This thesis considers business processes as a market-centred description of an organisation's activities, implemented as information processes and/or material processes. A business process is considered to be engineered to fulfil a business task, and involves a mix of coordinated information and material workflows.

2.1.1 Business Process Challenges

Modern enterprises exist within complex networks which need to adapt to a constantly changing environment. The architecture of an enterprise is the organizing logic of the individual business processes which make up its operations [250]. In the case of industries which create complex products or provide complex services, such as producing highly engineered products or addressing extremely dynamic customer needs, the strategy and consequently the business processes of these enterprises often exhibits a high degree of complexity. In these environments creation of dependable, yet efficient, business processes poses a significant challenge.

Processes are executed within networks of other processes and interact with these external processes. This synchronisation between processes effectively defines a combined process, often of far greater complexity than the individual component processes. This behaviour, however, is essential in business processes, as the value which is normally added by a business process arises primarily through the coordination of separate contributions, for example in organizing multiple contributions to the preparation of a quotation. This complexity makes traditional manual verification difficult and error prone, with a classic example being the failure of business processes in the Barings Bank debate [158]. The 233-year-old UK Barings Bank went bankrupt in February, 1994 after it sustained losses of $1.4 billion, incurred in a matter of days by a single young futures trader, Nicholas Leeson, in the Singapore branch. Inadequate process controls and other business process failures meant that Leeson's unauthorized futures trading went undetected by the headquarters until the very end.

Real-world processes add further complexity because processes may interact with elements outside the processes' control and whose behaviour is unknown. This motivates the need for probabilistic decision points in the modelling of business processes; because the outcomes of complex decisions within a process can appear random and/or not possible to predict in advance. This complexity is an inevitable consequence of business processes' existence in the physical world where many material processes, such as casting metal or performing a measurement, ultimately rest upon quantum mechanics and have an inherently stochastic nature. In such a stochastic system, characterising properties such as the number of items that remain in a queue or the resources consumed per job done will require answers in terms of probabilities. For example, a business process which consumes various resources in a metal casting process has key stochastic characteristics such as the mean consumption of resources per item cast.

Many business processes make use of various limited resources such as stock levels, time, or available liquidity, and the accurate provisioning of these resources is the key to their safe and efficient execution. An essential activity in efforts to develop such systems that integrate with their environments is constructing formal models of the processes involved. Currently, tools that support such activities simply allow for modelling such systems, with even the most sophisticated just performing basic simulation of the execution of business processes to determine future stock levels. Simulation, however, does not provide verification, and the properties which can be analysed, and the accuracy of the analysis, are limited.

The predominant approach to developing business processes is firstly to develop conceptual process models, which are used for development of an implementation, typically involving a mix of automated and manual processes. This approach normally leads to a design where the process is analysed to see if it meets its

goals, and any adjustments needed can be done before a costly implementation. Typically the design is desired to conform with complex categories of properties such as:

- **Validation properties:** testing whether the business process behaves as expected in a given context; e.g. does this financial process comply with the Sarbanes–Oxley act?
- **Performance properties:** evaluating the ability to meet requirements with respect to throughput times, service levels, and resource utilization or other quantitative factors; e.g. how long does it take to deliver a package?
- **Stochastic properties:** evaluating the probability of behaviour within the system; e.g. is there a greater than 98% chance the package will be delivered in 3 days?

There is substantial evidence [99], [108], [283] that being able to determine such properties and their exact quantitative values early in the design phase can result in smoother integration, accurate provisioning to meet service level agreements, and significant cost savings. Safety critical processes frequently combine the need for proof that safety properties are kept while simultaneously requiring a specific performance profile. Many business processes make use of various limited resources such as stock levels, or time, and the accurate provisioning of these resources can increase their safe and efficient execution.

The improvement of business processes is another task that is typically done by hand, where improved configurations are found by a process of trial and error, often taking many years to arrive at an optimal practice. In many cases, determining the optimal way to restructure a process, for example due to changing market conditions such as the introduction of new disruptive technology [112], or the merger of two companies, is unclear. The naive approach of simply replacing one function in an established business process with a new technology does not usually lead to maximal gains. This approach also means that the potential opportunities such changes in business processes can provide, for example the possibility of creating radically new ways of doing business, is often overlooked.

Being the first to fully realise the beneficial possibilities of these advances, and adopting the radically improved business processes which they allow, can be the key to a competitive advantage. For example, Toyota is widely viewed as being the first to truly realise the revolutionary benefits of IT-based inventory control and the consequent *Just-In-Time production* workflows it enabled [263], [288]. However, it is arguable that the technologies that enabled this advance were in place several decades before.

Finally, the trend towards using *visual diagrams* to model business processes has mostly neglected formal models which would allow effective quantitative analysis [278]. Instead, this trend means that analysts tend to focus solely on the qualitative behaviour of business processes.

2.2 Business Process Management

In recent years the adoption of a range of different process management approaches such as Lean Management [288], Activity-based Costing [61], [273], Total Quality Management [146], Business Process Re-engineering [126] and Process Innovation [87] has culminated in the approach of Business Process Management (BPM). Arising from a view of business processes as a set of competitive assets to be managed, it is a discipline that combines knowledge from information technology and from the management sciences and applies this to operational business processes [19], [276]. Determining the right level of investment in the resources needed, proper performance monitoring of the process, sound maintenance of the process, and management of the process life cycle can maximize operational performance.

This field has received considerable attention in recent years due to its potential for significantly increasing productivity and decreasing costs. However, perhaps because it is cross-disciplinary, BPM practice and research is fraught with a lack of universal terminologies [19], [134], [185] and terms are used loosely to represent distinct scope and feature differences [134]. Moreover, today there is an abundance of BPM systems, which are generic software systems that are driven by explicit process designs to enact and manage operational business processes [276], but all employ slightly different approaches. However, its leading proponent van der Aalst [276], has advocated for a consensus that business process management is defined as:

> *Supporting business processes using methods, techniques and software to design, enact, control and analyse operational processes involving humans, organizations, applications, documents and other sources of information.*

The specific details of how the BPM approach is implemented in organisations vary widely, but in general, the main stages are:

1. **Goal Identification:** A specific aim for the set of business processes to be managed within an organisation is identified. Ideally, these goals should be defined so as to be measurable such that the effect of changes to the

underlying business processes can be expressed in terms of the formulated goals. Crucially, this step involves the formulation of a set of *business goals*, e.g. "Improve delivery times" or "reduce medication errors". This stage is a manual step and typically involves meetings and discussions in order for a business analyst to fully grasp the requirements of the BPM process.

2. **Business process modelling:** Once a clear set of goals has been formulated, a model is constructed using a business process modelling language. The purpose of this stage is to design processes which meet the defined goals. BPM suggests a number of approaches to doing this, but normally this step is achieved by manual modification.

3. **Implementation:** In this stage the designed processes are implemented within the organisation. A key feature of a number of process modelling languages is that they allow for automatic translation into executable code which manages the business process. For example, in the case of Business Process Model and Notation (BPMN) diagrams, tools allow the translation into executable *web service business process execution language* (WS-BPEL) [211] code, which defines web services employed to perform a business process.

The BPM paradigm evolved from Hammer's [126] and Davenport's [87] Business Process Re-Engineering (BPR) concept of process innovation through radical change of existing business processes. However, it became apparent that business processes could not be designed to work correctly the first time. Therefore, as BPM was developed as a process of iterative, and incremental adjustment of business processes became its focus. The fine-tuning of business processes is done through the *BPM life cycle* which, as defined by van der Aalst [19], involves three key activities illustrated in Figure 2.1.

In Figure 2.1 the *(re)design* phase, a process model is designed. This model is transformed into a running system in the *implementation/configuration* phase. If the model is already formalised, this phase may be very short; however, in the case of an informal model this may require substantial effort. After the system supports the designed processes, the *run & adjust* phase starts. The process is not redesigned and no new software is created; only predefined controls are used to adapt or reconfigure the process.

Central to the adjustment of a process is the analysis of an existing process. Figure 2.1 illustrates in which phases the two standard types of business process analysis occur. In standard BPM, while the system is running, event data is

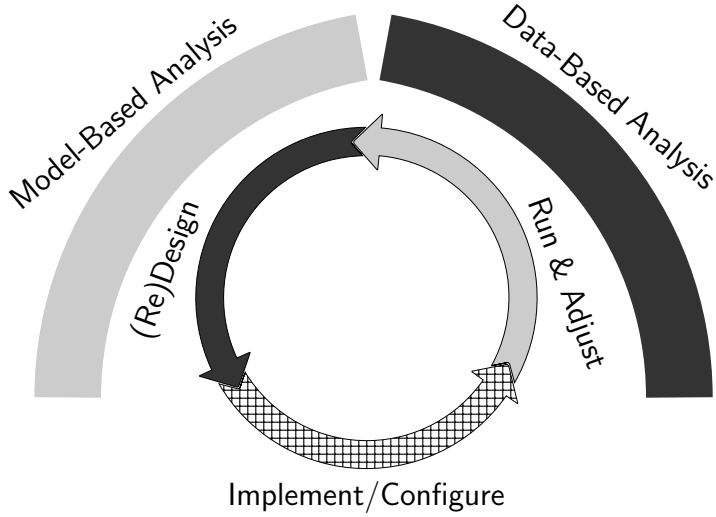


Figure 2.1: *The BPM development approach system life-cycle (source [19]).*

collected, and this data is used as the basis for process improvement, e.g., to discover bottlenecks, waste, and deviations. This is input for the redesign phase, where a process is then manually altered to account for these issues.

A classic example of the benefits BPM aims to provide is described by Ko [159]:

> *Consider the work of a typical design engineer in a company. We cannot measure value-added productivity just by looking at what he or she does. One has to follow the flow of information as the design evolves into the finished product. At certain points in time, tests or analyses are conducted that help engineers make decisions. The trouble is that these results of tests and analyses sit and wait in an information warehouse (inventory) until someone picks them up. Following this, the results could go through several more people and departments, adding to the delay. One can easily see that the problem is not unlike traditional batch-and-queue manufacturing, and that the answer is flow.*
>
> *The ideal workflow, processing a customer order as though it were the only order, pre-supposes a continuous flow of information and materials. Although a one-piece order is idealistic, small lots are not. In small lot manufacturing, by keeping the processes close together, the materials keep moving, and waste is minimized. Toyota*

> *identified seven non-value-adding wastes: Over-production, Waiting, Unnecessary transport, Excess inventory, Defects, Unused employee creativity and Motion.*
> *As a result, no one produces anything before it is needed by the next person or step in the process (i.e., no waiting, minimum overproduction and transport/movement). Where idealized one-piece flow is not possible, inventory buffers are judiciously introduced (no excess inventory). This is Toyota's secret, which enables its engineers to make a car in one year, when its competitors take two.*

2.3 Describing Business Processes

Although in smaller enterprises business processes may be ad hoc and never run the same way twice, once an organisation has reached sufficient size it is common practice to attempt to formally define the steps taken to achieve a given objective. Typically, companies will formally define their business processes when a process is repeated and of high value. Processes such as insurance claims handling, loan processing, or manufacturing tend to fall into this category. Formally describing business processes saves companies time and money, and running a process the same way each time maintains quality levels, as well as safety guarantees and can ensure regulatory compliance. It is a central step in the application of BPM methods.

A number of different abstract representations of business processes have been proposed. The construction of these representations has come to be known as *business process modelling*. In this approach, organisations are encouraged to think in processes instead of functions and procedures. The idea is to look at chains of actions in the organisations as opposed to individual actions, with the key element being the sequencing and control flow of actions.

The term *business process modelling* itself was coined in the 1960s in the field of systems engineering by Williams in his 1967 article *Business process modelling improves administrative control* [285] His idea was that techniques for obtaining a better understanding of physical control systems could be used in a similar way for business processes. However, the underlying techniques of business process modelling themselves are commonly considered to have originated from Gilbreth's seminal 1921 paper [113] *Process Charts - first steps to finding the one best way to do work* which is credited with developing the first method for formally documenting process flow. Considerable development has since taken place which allows additional features to be added to these models, crucially data flow [32] and various logics for complex branching [183].

Business process modelling is useful for three basic reasons, which may in turn support several business goals [213]:

1. **Describing a process:** A process is modelled so as to describe it. We could have different target audiences for these descriptions, for instance, humans, in which case understandability is important [213], or machines, in which case formality is important.
2. **Analysing a process:** Process analysis consists of assessing the properties of a process. Process re-engineering and improvement relies on an analysis of existing processes to identify redundant or sub-optimal steps. A formally described process can automatically be verified with regard to both structural properties such as coupling and cohesion [221] and dynamic properties such as the absence of deadlocks, liveness properties, etc.
3. **Enacting a process:** Ultimately the purpose of a business process is to be enacted, for simulation purposes or to provide some level of support for process execution. This support can take different forms: reacting to events triggered by the execution of the process, checking that specific constraints are satisfied or driving the execution of the process [84]. Again a formal description is required to ensure that an unambiguous translation can be made from a process model to enacted process.

2.3.1 Classifying Business Process Modelling Languages

There is a large and diverse range of different approaches to modelling business processes, Vergidis, Tiwari and Majeed propose a classification of the main approaches. This classification considers their structural characteristics and their capabilities for analysis and optimisation [278]. These approaches fall into one of three main categories shown in Figure 2.2 with key representative examples is shown in each of the sets.

Diagrammatic models involve business process models that sketch a business process using visual diagrams. These are derived directly from Gilbreth's original flowchart concept [113]. These simplistic diagrams depict a business process, frequently without employing a specific standardised notation and are useful for fast and informal process representation, but they lack the necessary semantics to support complex constructs. Since then, the main contribution in this area has been to standardise the notation of these models as typified by the Unified Modelling Language (UML).

Business process modelling benefited from standardized diagrammatic approaches as this approach was still relatively easy to use. However, these models can be misinterpreted as there is no way to ensure consistency across models, and they

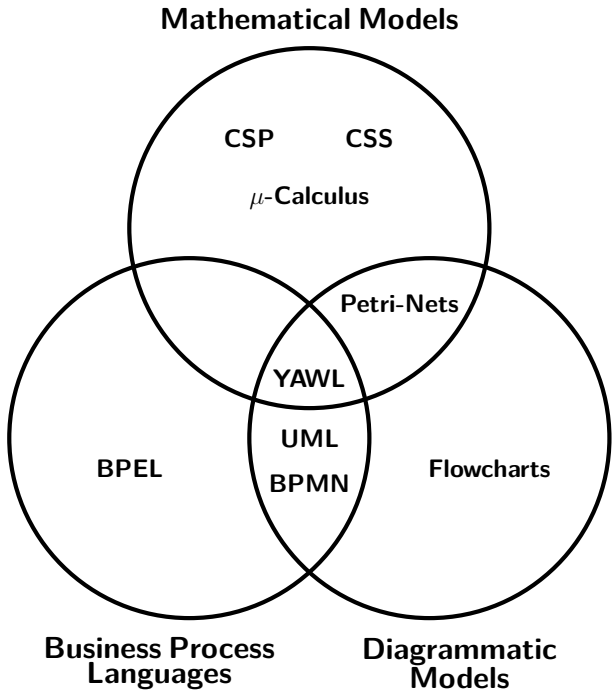


Figure 2.2: *Classification of the current main business process modelling languages.*

can not be translated into executable systems, or rigorously analysed. Typically, analysis consists solely of the inspection of diagrams and the conclusions are heavily dependent upon the skills of the analyst.

Mathematical models correspond to models in which all the elements have a formal mathematical underpinning. In their purest form these approaches are in essence derived directly from the ideas presented by Tony Hoare in his 1978 paper *Communicating sequential processes* [137], [138], and are effectively algebraic systems for reasoning about concurrent systems. An advantage of formal models is that they can be verified mathematically, and can be checked for consistency and other properties. However, their complexity does not make them usable within the business community where only highly skilled analysts can comprehend these descriptions.

Petri-nets [220] are an example of a formal modelling technique that combines a visual representation using standard notation with an underlying mathematical representation, but even this involves considerable complexity and has not seen

widespread adoption in the business community. In general, building a formal business process model can prove much more complex and demanding compared to purely diagrammatic techniques, and this has limited the adoption of using mathematical models for the modelling of business processes.

Finally, the third category is *business process languages* which specifically support business process modelling and frequently also support process execution. Here the modelling formalism is developed expressly to describe business processes and typically attempt a degree of formalisation in an attempt to retain the consistency and potential for further analysis of mathematical models, but reduce the complexity. These are context-specific languages aimed mostly at modelling business processes and are usually designed so that an executable system can be generated from them.

2.3.2 The Main Business Process Languages

The different mathematical models presented in the classification of approaches to modelling business processes, shown in Figure 2.2, are Turing complete and as such each language has the same formal expressibility. Diagrammatic models without a formal basis, while often appealing in their simplicity and often widely used in practice, do not allow for automatic verification or automated enactment of processes as their precise meaning is left undefined. Business process languages seek to bridge this gap and aim to allow, to varying degrees, both the description, analysis and enactment of business processes.

Mathematical models are clearly capable of describing business processes but these general formalisms are challenging to employ. Dedicated business process languages are predominantly developed with a focus on *suitability* [154] which is an informal notion of expressiveness which takes the modelling effort into account. Frequently the main approach to reduce the modelling effort required is to allow modelling by diagrammatic means and has come to dominate practical business process modelling. Today (2013) the dominant business processes languages are: UML, BPMN, and Yet Another Workflow Language (YAWL), all of which have a graphical notation. Furthermore, these languages frequently claim to have a formal basis and therefore allow for automated analysis and enactment of business processes.

2.3.2.1 Unified Modelling Language (UML)

UML [208] was originally developed by Booch, Jacobson and Rumbaugh at Rational Software in the 1990s [125]. UML become managed by the Object Management Group (OMG) in 1997, and is still managed by this group today. In 2000 the UML was accepted by the International Organization for Standardization (ISO) as industry standard for modelling software-intensive systems and is today standardized according to ISO/IEC. The current version is UML 2.4.1 which was published by the OMG in August 2011.

UML has become a standard language for the modelling of software requirements and design. UML includes a set of graphic notation techniques to create visual models of object-oriented software-intensive systems which offer a standard way to visualize a system's architectural blueprints. It combines techniques from data modelling (entity relationship diagrams), business modelling (work flows), object modelling, and component modelling. While primarily designed for modelling software systems, it can also be used for business process modelling. Business process modelling is primarily done by constructing *activity diagrams* which visualize workflows as sequences of actions to be performed including control flow and data flow. The semantics of business processes defined in this fashion are based on token flow and tokens may also be employed to model data flow. Organizational structures can be accounted for by means of *class diagrams* and these can in turn be related back to the activity diagrams which capture workflow by means of *activity partition* and *swim lane* mechanisms.

Business process models built in UML are not executable and are mostly used for design, analysis, or documentation purposes. However, UML models may support process enactment by functioning as a set of requirements for development of specific atomic actions which form part of a business process. However, UML has in all versions lacked a precise semantics [101], [243] which hinders both enactment and rigorous analysis of models. In addition it should be noted that UML is an extremely verbose and complex standard and that in all versions a large number of practical problems in employing this approach have been encountered [106], [264].

UML's lack of semantic precision has been addressed by many researchers. A first approach to the formalisation of UML was done by Lilius and Paltor [175]. This begins with a formalisation of the structure of a UML state machine as a tree of states. They employ this formalisation for the construction of a transition selection algorithm that enables a system to run to completion. This first approach provides a foundation for many future developments, but is limited by a requirement that systems run to completion to enable their formalisation. In the case when a deadlock or race condition occurs, naive

application of this formalisation approach is not possible. Work by Eshuis [98] performs formalisation by means of hyper-graphs and allows the imposition of semantics by the model checker NuSMV [74]. In this work, the focus is on refinement where the verification of requirements for a UML model, expressed as a simpler UML model, is confirmed by NuSMV. This excellent work produces a sound formalisation amenable to formal verification however the treatment of rewards and stochastic behaviour is not accounted for and not readily supported by the underlying NuSMV tool.

Accounting for quantitative properties of UML models was initiated by Canevet et. al. [66] who employed the Performance Evaluation Process Algebra (PEPA) [135] stochastic process algebra as an intermediate language in the performance analysis process, and this dictates the semantics imposed on UML. This approach allow for quantitative verification but is limited by the performance of the PEPA tool. Improved quantitative verification which also allows for modelling stochastic properties is developed by Jansen et. al. [148]. This work formalises UML Statecharts which are extended with probabilities and adds data (in the form of rewards) to a UML model. This framework makes it possible to analyse the resources consumed by a system.

2.3.2.2 Business Process Model and Notation (BPMN)

BPMN [207] has been developed as a dedicated language for the modelling of business processes. Designed by the Business Process Management Initiative (BPMI) which released BPMN 1.0 in May 2004. BPMI merged with the OMG in 2005 and this organisation has since maintained it, releasing the current version 2.0 of BPMN in 2011.

BPMN is intended to be a standard for business process modelling that provides a graphical notation for specifying business processes in the form of Business Process Diagram (BPD), based on a flowcharting technique similar to activity diagrams in the UML. The objective of BPMN is to support business process management, for both technical users and business users, by providing a notation that is intuitive to business users, yet able to represent complex process semantics. As such the The primary goal of BPMN is to provide a standard notation readily understandable by all business stakeholders. The notation inherits and combines elements from a number of previously proposed notations for business process modelling, including the XML Process Definition Language (XPDL) [291] and the Activity Diagrams component of the UML [208].

The BPMN language itself is extremely circuitous, and contains 116 inter-definable constructs. These cover a wide range of cases but most practical use of BPMN does not make use of the vast majority of elements. In fact a large study of real-world BPMN usage [193], [238] found that "the average BPMN model uses $< 20\%$ of the available vocabulary" and more than 70% of models surveyed consisted of only eight core elements. Instead of defining a core of independent constructs in terms of which other constructs can be defined, as demonstrated for BPMN 1.0 in [56]. The fuzzy overlapping of different constructs prevents clear descriptions of individual constructs in one place within the standard and makes their comprehension unnecessarily complicated by forcing the reader to simultaneously and repeatedly consider multiple sections of the standard document.

In practice the standard, as formulated in [207], appears not to be implementable due to the numerous details of its execution semantics which are expressed with insufficient precision. Indeed the BPMN standard explicitly states that "the execution semantics are described informally (textually), and this is based on prior research involving the formalization of execution semantics using mathematical formalisms." [207, page 445]. The shortcomings of these semantics have been observed, for all BPMN versions, by various authors [54]–[56], [72], [88], [92], [120], [193], [238], [240], [242], [280], [287], [289], [295] (this list is not exhaustive).

This lack of semantic precision has led to a diversity of implementations of BPMN concepts which typically realize only subsets of the standard and which are often only partially compatible with each other. The BPMN specification includes an informal and partial mapping [207, page 445] from BPMN to Business Process Execution Language (BPEL) 1.1 [211]. A more detailed mapping of BPMN to BPEL, such as [215], has been implemented in a number of tools. However, the development of these tools has exposed fundamental differences between BPMN and BPEL [242], which make it very difficult, and in some cases impossible, to generate human-readable BPEL code from BPMN models. Even more difficult is the problem of BPMN-to-BPEL round-trip engineering: generating BPEL code from BPMN diagrams and maintaining the original BPMN model and the generated BPEL code synchronized, in the sense that any modification to one is propagated to the other.

Fundamentally BPMN lacks a general notion of state and as a consequence, the specification of relevant data-dependent execution conditions are only poorly supported. This makes the data management of an executable BPMN version compiler-dependent and not portable. For example, data objects, which are associated with activities or sequence flow, are used only informally, in particular with underspecified assumptions on the input/output selection at task nodes or sequence flows. This additionally makes any sort of quantitative formal analysis

of standard BPMN impossible. However, the standardisation of BPMN has lead to its widespread use by business practitioners and consequently there have been a number of attempts to provide a formalisation of its semantics.

Wong and Gibbons [289] take an abstraction refinement approach to the formalisation of a subset of BPMN models as Communicating Sequential Processes (CSP) process-algebraic expressions. Their work expresses the syntax of BPMN in the Z notation [267]. They proceed to define a semantic function which takes a syntactic description of a BPMN diagram and returns a CSP process that models the behaviour of that diagram as the parallel composition of processes corresponding to the states of the diagram. The semantics they impose abstracts the internal flow of individual task states, and models the sequence of task initialisations and terminations within a business process. This work is extremely thorough with regard to constructing a formal translation from BPMN to CSP, and the specific semantic interpretation forced upon BPMN models and the resulting CSP models produced are very circuitous. While they are theoretically amenable to formal analysis, in practice the verbosity of the generated models makes it computationally expensive to perform analysis. Probabilistic properties and rewards are not supported, and while variants of CSP exist that support these features, considerable reworking of their approach would be needed to account for these.

Work by Ouyang et al. [215], presents an approach to BPMN formalisation by mapping BPMN's graph oriented structure to BPEL, which is a mainly block-structured language. This approach focuses on dealing with the problem that graph-oriented process definition languages allow parallelism and other constructs such as deferred choice which can complicate their mapping to a block structure. In a similar way to Wong and Gibbon's approach, the mapping which was developed focuses on the sequencing of tasks and functions by decomposing a BPMN model into *components* and mapping these to fixed blocks of BPEL code. While this work focuses on BPMN to BPEL conversion, the approach is intended to be generalised to the mapping of any graph-based language to a block-based language. Although this approach is readily comprehensible, the lack of a full decomposition down to individual BPMN elements does limit the fit between BPMN models and the generated BPEL models, which frequently become digressive as a result. The overall approach, however, is excellent, and of particular note is that this approach avoids imposing further semantic interpretation on BPMN by focusing solely on mapping to BPEL and allowing the semantics of BPEL to determine the execution of a BPMN model. However, stochastic elements and rewards are not addressed.

Dijkmana and Dumas, working with Ouyang [88], have repeated this control-flow focused approach in developing a mapping from BPMN to Petri-nets. Akin to the previously described work of Ouyang, this also takes a component-wise

approach to mapping, with similar limitations. A key contribution here is the comprehensive identification of specific areas where the BPMN specification is lacking. However, a limitation of this approach is the restriction that the mapping is to plain Petri-nets, due to the need to clearly define source components for translation. Jian-Hong et. al. [295] attempt to extend their work to perform translation into YAWL and this allows for more flexible synchronization between components. However, in both cases, the limitations on the nets prevent the addition of rewards and stochastic behaviour and are poorly suited to capturing real-world processes.

An approach by Prandi et al. [230] does provide for BPMN models which support rewards and probabilistic elements. This effort involves conversion of BPMN models into a model expressed in the Calculus for Orchestration of Web Services (COWS) [173], which is converted into a model that can be analysed using a model checker. It suffers, however, from poorly defined semantics for the translation from BPMN to COWS, and many corner cases are left undefined.

An approach by Nicolae et. al.[200] attempts to formalise BPMN by mapping, via a pattern matching approach, to UML models. Additionally, the Object Management Group has in June 2013 released a *UML Profile for BPMN 2 Processes* [209], which provides a "conceptual mapping" from BPMN models to UML models. While these mappings highlight the correspondence between UML and BPMN and do improve the possibilities for the interchange between users of business process models, they do not provide semantic clarity to UML (described in Section 2.3.2.1).

2.3.2.3 Yet Another Workflow Language (YAWL)

YAWL [23] is a workflow language developed by Hofstede, and van der Aalst, in 2005. This business process modelling language is intended to be a language which exhibits a large degree of *suitability* [154] with a set of 126 workflow patterns [24] also collected by Hofstede and van der Aalst. A workflow pattern is defined as "the abstraction from a concrete form which keeps recurring in specific non-arbitrary contexts" [246]. As such, the YAWL language is a "reference implementation" [139, page 11] of these workflow patterns. The language is supported by a software system [20] which is available as Open Source software.

Observing that Petri nets [220] came close to supporting most of the Workflow Patterns, the designers of YAWL decided to take Petri nets as a starting point and to extend this formalism with three main constructs, namely *inclusive OR-joins*, *cancellation sets*, and *multi-instance activities*. These three concepts are aimed at supporting five of the collected workflow patterns that were not already

directly supported in Petri nets, namely *synchronizing merges*, the *discriminator*, *N-out-of-M joins*, *multiple instance with no a priori runtime knowledge* and *cancel case*. In addition, YAWL adds some syntactical elements to Petri nets in order to more *suitably* capture other workflow patterns such as *simple choice* (XOR-split), *simple merge* (XOR-join), and *multiple choice* (OR-split). Some of the extensions that were added to Petri nets to achieve this, were difficult or even impossible to re-encode back into plain Petri nets [23]. As a result, the original formal semantics of YAWL is defined as a Labelled transition system and not in terms of Petri nets.

The motivation for the choice of constructs in the YAWL language is driven by a of collection workflow patterns [24]. However, the selection of patterns seem to be chosen without a deductive or statistical underpinning to validate the pattern selection, indeed it is conceded that for the original 20 workflow patterns "the selection of these patterns was done in an ad-hoc manner and the description of the patterns in natural language has been rather ambiguous" [196, page 1]. Indeed without a set of clear criteria for what qualifies as a pattern a steady growth in the underlying workflow patterns can be observed starting with 20 workflow patterns in 2003 [24], reaching 43 patterns in 2006 [255] and in 2010 expanding to 126 patterns (including data and resource perspectives) [139]. Additionally, a number of patterns are not of a fundamental character but can instead be constructed from a smaller set of simpler patterns, for example it has been shown that the 43 patterns from [255] can be defined by parametrizing 8 precisely defined abstract models which represent well-known, basis, sequential or concurrent programming constructs [53]. It should be noted that the workflow patterns as presented in [24] are described using natural language and lack unambiguous definition. The "rather ambitious" character is conceded in [196, page 1] and given as motivation for formalisation of the patterns which this paper admittedly leaves "incomplete".

YAWL can be seen as another attempt to formalise this set of workflow patterns by constructing a semantics based upon coloured petri-nets [149]. Given that YAWL is based upon Petri-nets, the foundational paper [23] does succeed in providing a formal mathematical basis for modelling the control-flow perspective (a perspective abstracted from data and communication) of business processes. Later work has extended YAWL in the form of *newYAWL* [252], [254] to also cover data and resource perspectives of a business process. This work has allowed for the semantics of YAWL to be formally defined by means of Coloured Petri Net [294]. However, this formalisation is extremely complex with the resultant Coloured Petri net model for newYAWL's semantics being "55 distinct pages of CPN diagrams and encompasses 480 places, 138 transitions and in excess of 1500 lines of ML code" [252, page 17]. A semantic model of this size has

some limitations and "Perhaps the most significant of these is that the scale and complexity of the model obviates any serious attempts at verification" [252, page 18].

2.4 Requirements for a Business Process Language

Fundamentally, a business process language must describe the control-flow structure of a business process. The overall requirements determined by the industrial partner are described in 1.2 as Objective 1. Further discussion of the objective led to the industrial partner developing a list of modelling language requirements shown in table 2.1. The motivation for these additional requirements in a medical context are discussed below.

Summary of requirements for a BPM language for IHS	
Requirement 1	Be able to model non-deterministic control flow alongside probabilistic control flow.
Requirement 2	Able to model the use of medical resources, including more general quantities such as time and money.
Requirement 3	Usable with limited technical training which is able to abstract the essential elements of a medical workflow in a concise manner.
Requirement 4	Amiable to formal mathematical verification such that errors can be detected at design time.

Table 2.1: *Core BPMN Requirements*

Medical workflows of the type described in the motivating context Section 1.1 involve the description of both people, medical devices and their interactions. These frequently exhibit a well-defined control-flow structure. Medical activities often involve performing actions where a range of possible responses can be expected from the patient, frequently due to the vast variation and complexity of human physiology. Medical knowledge is therefore frequently expressed in probabilistic terms, e.g. cancer of form X when treated with form Y has a 20% chance of remission, 25% chance of death and a remaining 55% chance of being ineffective. When no probabilities are known, a probabilistic process can be

viewed as being a non-deterministic process. Therefore probabilistic models have an expressive limit as non-deterministic models, this is frequently the case in medicine probabilities of specific outcomes are yet to be determined or may be invalid in a specific context. In addition the human element in practising medicine is inherently non-deterministic and actors involved in these processes will make choices which cannot be determined before execution of the process. This is the motivation behind Objective 1a of this thesis, and requires a formalism which is able to model both non-determinism and probabilistic choice.

Medical workflows also make use of valuable resources which due to legal and financial constraints must be carefully accounted for. The variety of resources consumed ranges from tangible assesses such as drugs and costs to more abstract notions such as time and quality. This gives rise to Objective 1b which requires a mechanism which can account for arbitrary numerical data throughout the process.

In addition, the designers of these workflows will frequently be medical administrators without formal training in mathematical analysis techniques and therefore a method is required that allows for the easy description of workflows without excessive notation or complex structural semantic rules. This requires that the language has a small number of constructs and a straightforward notation. Ideally this notation should be graphical and seem familiar to practitioners with some knowledge of business process modelling. Objective 1c expresses this requirement. Furthermore, a suitable level of abstraction must be chosen as the specific implementation of medical workflow may vary between industry and even within the same medical environment a wide range of different implementations may exist, e.g. in one case messages may be sent electronically between actors in a medical workflow and in another environment messages may take the form of face-to-face communication. Employing the workflow patterns of [24] as the base for choosing constructs for a modelling language lacks a deductive basis for their selection. In this work the statistical study of [193] informs the choice of elements to include in a modelling language. The overall idea is to employ the elements determined to be the most commonly used in [193]. Complex models are then built by composing these simple elements to build larger structures. The aim being to allow development of libraries of common medical workflows each composed of simple base elements.

Medical workflows frequently have high-stakes outcomes and therefore it is crucial they are understood already at design time, e.g. when designing a medical robot it is desirable to determine the expected impact it will have on an existing or new workflow before it is deployed. Given that workflows may execute for a long time and may take actions that cannot be undone in a simple manner, it is essential to detect errors at design time. Therefore the chosen formalism should permit processes to be formally analysed, optimised

and readily employed and communicated by business practitioners. This implies that a formalism is needed which has a sound semantic basis, and of the main established business process languages none have a workable formal semantics which enable automatic verification. Consequently, it is not clear how the set of possible processes allowed by each of these languages should be unambiguously interpreted and therefore automatic verification cannot be performed. While YAWL does have a formal semantics it is of little use to a practitioner who wishes to understand the semantics of YAWL as it is almost impossible to consult the enormous semantics [294] to answer questions about the language and it does not allow formal verification.

The various attempts to provide these established business process languages with a formal semantics discussed in Sections 2.3.2.1 to 2.3.2.3 all lack support for practical verification which takes advantage of the advanced features of the current theory of process models and making these applicable to formal analysis and verification tools. Many formalisations, such as [200], [209], [215] simply provide mappings between one business process modelling language and another. While those that are based on Petri-nets such as in [88] or [252] have limited tool support, with the state-of-art of these tools [29], [281] only allowing specification of properties using CTL or LTL-like logics and not accounting for stochastic or quantitative properties.

Of particular note is the approach by Prandi et al. [230] to translating BPMN processes into stochastic COWS models, which does allow for quantitative analysis of BPMN via the powerful model checking tool PRISM [217]. However, the addition of stochastic and quantitative properties to the model are introduced only after translation into COWS and this prevents their addition by business practitioners working in BPMN .

UML has seen a number of complete tool chains produced which allow for formal analysis [66], [98] and even stochastic and quantitative properties have been accounted for by Jansen et. al. in [148]. However, the suitability of UML as a language for the modelling of business processes is debatable [54], [253]. Further, the focus of the formalisation efforts has been on the state-charts component of UML and not the suggested combination of activity and class diagrams suggested for business process modelling.

2.5 Chapter Summary

This chapter has discussed the main concepts of business processes and established a definition of them which fundamentally views business processes as *sets of connected activities*. Further, it has briefly described how the organisation of these activities defines the performance of an enterprise. The emerging approach of business process management and the state of the art of methods employed by this approach in the BPM life-cycle is outlined to place the later developments in context. A categorization and overview of the main approaches to business process modelling along with the strengths and limitations of current approaches in terms of support for formal verification and suitability for modelling medical workflows was presented. This analysis motivates the development of a business process modelling formalism presented in the following chapter.

Chapter 3

Modelling Business Processes

"To succeed in business it is necessary to make others see things as you see them." (Henry Ford 1919)

Overview

This chapter defines the syntax and static semantics of a formalisation for the modelling of business processes based on the requirements described in Section 2.4. This Chapter employs graphs as the starting mathematical concept from which to develop a modelling language for the description of the control-flow structure of a wide range of business processes. This structure is extended to account for the stochastic nature of real-world processes, in line with Objective 1a. The requirements for a business process modelling language described in section 2.4 also require the ability to account for numerical data, typically resources consumed during the execution of the business process. This requirement expressed by Objective 1b is met by defining quantitative data annotations for both specific states of the business process and transitions between them. Using Business Process Model and Notation (BPMN) as a basis, a Core fragment is defined and given a formal denotational semantics.

The formalisation of BPMN that this chapter describes was first presented in [11], with a refined version appearing in [14], with a final complete formulation being made in [12].

3.1 Approach

The focus of this chapter is to define a syntax, a denotational semantics and structural semantics for the description of business processes. Based on the notion that a business process can be understood as *sets of connected activities* a starting point of directed graphs is employed to develop a modelling language. These *process graphs* allow for the expression of the control-flow structure of business processes. Based upon this foundation, a fragment of the BPMN modelling language is used a basis to construct a formal business process modelling language, Core BPMN with a denotational semantics described in Section 3.5.

The central idea in this chapter is do develop a minimal set of constructs which can be combined to capture complex business workflows. This is motivated by the observation [193] that only a small amount of the constructs in BPMN are actually used by business practitioners when modelling languages. Moreover, studies by Mendling, Reijers and van der Aalst in 2010 [186] and by Mendling in 2009 [187] have established a set of rules for process modelling with a view to improving model quality and supporting formal verification. These rules known as the 7PMG guidelines [186] are shown in Table 3.1 and these have informed the

approach to the formalisation of BPMN presented in this chapter. In particular the decision to keep a minimal set of constructs (Guideline **7PMG-G1**) and the decision to exclude the OR control flow element (Guideline **7PMG-G5**).

7PMG Modelling Guidelines	
7PMG-G1	Use as few elements in the model as possible
7PMG-G2	Minimize the routing paths per element
7PMG-G3	Use one start and one end event
7PMG-G4	Model as structured as possible
7PMG-G5	Avoid OR routing elements
7PMG-G6	Use verb-object activity labels
7PMG-G7	Decompose a model with more than 50 elements

Table 3.1: *7PMG Modelling Guidelines [186]*

Extensions to the formalism of *process graphs* are then developed which allow for stochastic branching via a Markov decision process style semantics and data annotations on both states and transitions in the business process. Compositionality of models is achieved through a message passing synchronisation mechanism. This Chapter also shows how this *process graph* formalism can be instantiated to capture the established language of the BPMN [207].

The modelling of resources used during execution of a business process is developed as an additional extension to the concept of process graphs. This extension allows for positive real numbers to be associated with transitions or states in the process graph which in turn can be used to model the use of resources associated with the given task or transition. Note that depletion of resources can be captured by using a positive real number to express the resources used.

A method for the modular development of business processes is introduced in the form of a message passing semantics which allows for the invocation of separate process graphs. In essence this message passing serves simply to combine separate process graphs by means of a separate set of transitions and does not introduce a fundamentally different mechanism than the basic control-flow of an existing business process.

This chapter seeks to establish a formal syntax for a modelling language suited to describing the type of workflows introduced in Section 1.1. Therefore the notion of process graphs is exploited to describe a subset of the most commonly used constructs of the BPMN language. The choice of employing the core control-flow constructs from BPMN as opposed to other modelling notations is based on BPMN being an established standard by the Object Management Group (OMG); an international, open membership, not-for-profit computer industry standards consortium [207]. In addition, while no single modelling language has come to dominate real-world business process modelling, BPMN does appear to be employed in the largest share of the most widely used business process modelling tools [193], [194], [239], [241]. In the context of this thesis, it is also important that BPMN is a a modelling language for which there is specific medical industry support [237], [249] and which the industrial partner company was familiar with.

It should be stressed that this chapter seeks primarily to define a syntax for process graphs and their extensions to cover the core constructs of BPMN. A denotational semantics is presented in Section 3.5 as an aid to modelling business processes. However as this work is concerned with the analysis of business processes the execution semantics of models is imposed when they are analysed in Section 4.5.2 as the totality of possible executions.

3.2 Process Graphs

The main abstraction on which business processes will be modelled is *connected directed graphs*. Graphs represent the control flow structure of business processes where nodes represent tasks and edges indicate the order in which the individual tasks of a business process are performed. Here a task is an abstraction of a piece of work which is to be performed as part of a business process. Definition 3.1 captures this sequencing of tasks within a well-defined mathematical structure.

Definition 3.1 (Process Graph)
A process graph is a tuple $P = (\mathbf{N}, \boldsymbol{\mathcal{F}}, n_0)$ *where* $\mathbf{N}$ *is a set of nodes representing tasks, of which* $n_0 \in \mathbf{N}$ *is an initial state, and the flow relation* $\boldsymbol{\mathcal{F}} \subseteq \mathbf{N} \times \mathbf{N}$ *is a relation which expresses the ordering of tasks in a process.*

Definition 3.1 allows for the expression of the control flow in a business process as a *directed graph* consisting of nodes $\mathbf{N}$ and edges $\boldsymbol{\mathcal{F}}$. Nodes represent basic elements of a business process such as activities performed and decisions made. A unique node n_0 is defined as an initial state of a business process.

Flows are links between nodes which express the relationship of one node to another, typically expressing the simple ordering in which elements of a business process are performed, although they may also capture message passing. An example of a business process modelled as a process graph is shown in Figure 3.1. In this figure a process of inspecting a medical drug before use is modelled; note how it includes both non-determinism (as shown in the grey coloured node) and that flows may allow execution to jump to previous states thereby allowing *recursion*, as highlighted by the dotted flow in Figure 3.1.

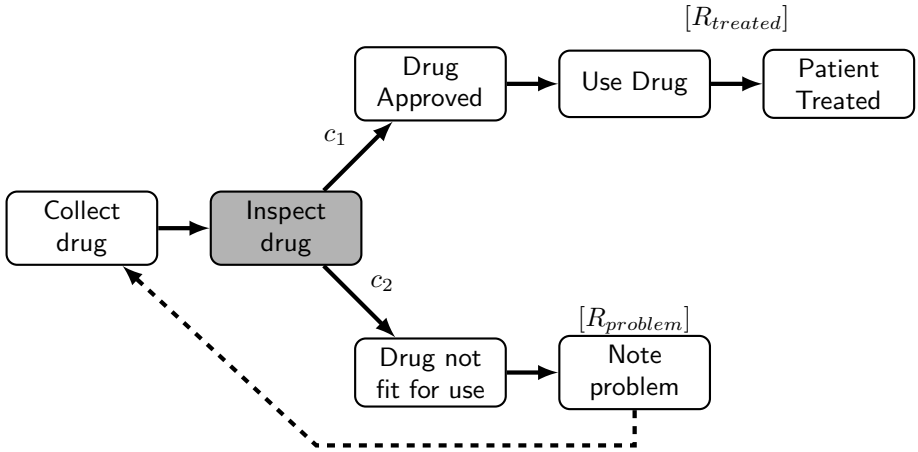


Figure 3.1: *Annotated process graph model of a simple medical process.*

To enable the formal analysis and the extension of this model of business processes, Definition 3.2 introduces a function which assigns labels to flows in a process model P.

Definition 3.2 (Flow Labelling)
Given a process graph $P = (\mathbf{N}, \boldsymbol{\mathcal{F}})$ *and* $\mathbf{L}$ *a set of unique labels,* $\mathsf{lab} : \boldsymbol{\mathcal{F}} \to \mathbf{L}$ *is a labelling function which assigns labels* $l \in \mathbf{L}$ *to flows* $f \in \boldsymbol{\mathcal{F}}$*, such that* $\mathsf{lab}(f) = l$.

For the example in Figure 3.1, the flows from the grey "Inspect drug" node are labelled c_1 and c_2 respectively.

To describe the sets of nodes connected by flows in the directed graph formed by P, the following definitions are employed:

Definition 3.3 (Successor Nodes)
Given a process graph $P = (\mathbf{N}, \boldsymbol{\mathcal{F}})$, the successor (output) nodes of $n \in \mathbf{N}$ are given by the function

$$\mathsf{out}(n) = \{x \in \mathbf{N} | n \boldsymbol{\mathcal{F}} x\}$$

Definition 3.4 (Predecessor Nodes)
Given a process graph $P = (\mathbf{N}, \boldsymbol{\mathcal{F}})$, the successor (input) nodes of $n \in \mathbf{N}$ are given by the function

$$\mathsf{in}(n) = \{x \in \mathbf{N} | x \boldsymbol{\mathcal{F}} n\}$$

3.2.1 Process Synchronisation

To manage the complexity of large business processes it is useful to construct the model by composing several smaller *sub-models*. Typically these sub-models will capture a specific role within an organisation but the decomposition can also be based on the location of operations or project phases. The central concept is that these processes orchestrate their individual actions so as to achieve the goal of the super-process, i.e. the larger business process which they are combined to describe. This approach is in line with the view of understanding the behaviour of organisations as activity systems [65].

A business process is essentially an abstraction of the actual work done in an organisation. Being able to compose a process from separate sub-processes greatly aids modelling because a large scale abstract model can first be built which coordinates separate processes, and later the individual sub-processes can be detailed. The complexity introduced by modelling large systems in this fashion rests on the coordination of these activities, a problem which is at the heart of informatics [191]. Further, the 7PMG guidelines [186] recommendation **7PMG-G7** suggests that models with more than 50 elements should be decomposed and as such a mechanism is required to achieve this.

Within process graphs, a natural approach is to enable orchestration by means of message passing. Here, messages are abstract notations associated with the transfer of any kind of signals between separate processes, and do not necessarily take the form of traditional messages. For example, a doctor receiving drugs from a pharmacy process could be modelled as the passing of a message between the pharmacy and doctor processes, as shown in Figure 3.2.

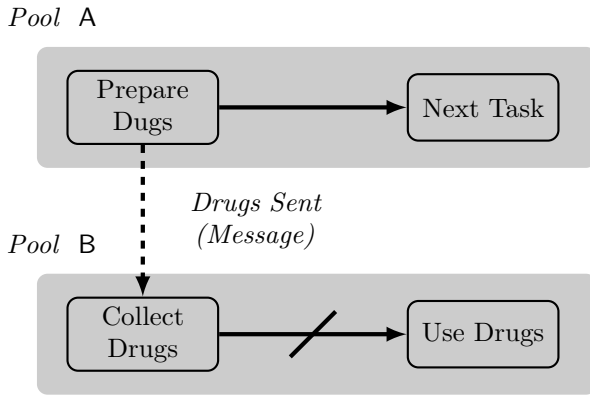


Figure 3.2: *Message Passing Example*

Many process calculi use message passing as an underlying mechanism for communication. For example, in the Calculus of Communicating Systems (CCS) [191] an *action* and a matching *co-action* together form a joined action, known as a *synchronisation*. Synchronisation in CCS is atomic and takes the form of a unique internal action that can be ignored by an external observer. Therefore, message passing is a synchronous event and for two communicating processes to proceed in their execution they must both be in a state where a message transmission (action) and message reception (co-action) are both enabled.

While CCS has extensively inspired the development of *process graphs* the synchronous communication of CCS is poorly suited to modelling business processes where messages may be sent to a different *process graphs*, modelling separate business processes, and a number of tasks may be performed after a message has been sent without requiring a return message. In practice most interacting business processes will both transmit and receive messages, however these events are independent and individually each takes the form shown in Figure 3.2.

In the case shown in Figure 3.2 the sending of drugs from business process *Pool* A to *Pool* B does not prevent *Pool* A from continuing execution. The receiving process, *Pool* B, requires the receipt of drugs before being able to proceed execution as indicated by the symbol between *Collect Drugs* and *Use Drugs* in Figure 3.2. However in *Pool* A messages are sent and execution may then proceed regardless of weather *Pool* B is ready to receive a message. Messages which are sent when the receiving process is not in a state waiting for a message are *lost*. Hence, if the receiving process later is in a state where a message must be received it must wait until a new message is sent.

This behaviour is a *partially-blocking asynchronous* means of communication where execution of a process halts when waiting for a message, but is able to continue execution when emitting a message. For example, in Figure 3.2 the tasks in *Pool* A may continue to execute regardless of the state of *Pool* B. This style of asynchronous synchronisation introduces *non-determinism* where the next event in the execution of a CCS model may be emission of a message followed by execution within the sending *process graph*, or the message may be received and proceeded by the execution of a task within the receiving process.

Formally Definition 3.5 introduces a message passing mechanism for process graphs:

Definition 3.5 (Synchronisation arc)
Given process graphs $P_1 = (\mathbf{N}_1, \boldsymbol{\mathcal{F}}_1)$ *and* $P_2 = (\mathbf{N}_2, \boldsymbol{\mathcal{F}}_2)$*, a synchronization arc is defined as a relation* $\mathcal{M} \subseteq \mathbf{N}_1 \times \mathbf{N}_2$ *which for* $n \in \mathbf{N}_1$*,* $m \in \mathbf{N}_2$ *and the relation* $n\mathcal{M}m$ *expresses that a message is passed from node* n *in process graph* P_1 *to node* m *in process graph* P_2*.*

In this chapter Definition 3.5 defines a syntactic element intended for the modelling of processes. In Section 3.5 an denotational semantic interpretation of message flows for an extension of process graphs will be formally developed in the form of a traces model allowing for the execution of communicating business processes. Further, in Section 4.5.2 the extended semantics imposed when performing an analysis of the business processes is discussed.

3.2.2 Stochastic Branching

Real-world business processes exhibit stochastic behaviour. This behaviour arises because of the underlying stochastic nature of physical processes, or the unpredictable nature of human behaviour. The source of non-determinism in real-world processes is frequently due to the fact that errors are possible, but some business processes may include inherently stochastic elements.

An example of basic non-determinism can be a process that involved a doctor administering a drug to the patient. Until the drug is prepared it is simply not possible to determine if the drug is fit for use. Only at the point when the drug is prepared would a decision be made, and in each case the flow would branch with one set of actions done in the case of a good drug, and one in the case of a bad drug. This type of non-determinism is shown in Figure 3.1 where the *inspect drug* node (marked in grey) is linked by the flow relation $\boldsymbol{\mathcal{F}}$ to more than

one subsequent node. Here, the flow of the business process depends on some external condition, c_1 (the doctor determines that the drug is ok) or c_2 (the doctor determines the drug is not fit for use), a decision which is not under the control of the process. Crucially, this non-deterministic behaviour may result in *recursion* where the flow of a process returns to an earlier state, as illustrated by the dotted line in Figure 3.1. The recursive nature of business processes is a key source of their complexity and makes their analysis challenging.

In this case, however, a second level of complexity is introduced because the doctor may have made a mistake in his determination of the condition of the drug. Typically, in a business process it is necessary to capture this type of uncertainty, and this motivates the introduction of *intention preserving* stochastic decision points where a decision is made which will have an unknown outcome from a limited range of outcomes. Including this behaviour in models of business processes is essential to be able to design processes which are efficient in the real world.

This need can be addressed through probability annotations (l, p) at decision points in a process graph, which dictate that the choice of transition from a decision point is made by first non-deterministically selecting a label l, and then performing a probabilistic choice according to different probabilities p associated with a specific label l. This behaviour preserves the choice of the actor in the business process while also allowing for stochastic behaviour, and is illustrated in Figure 3.3.

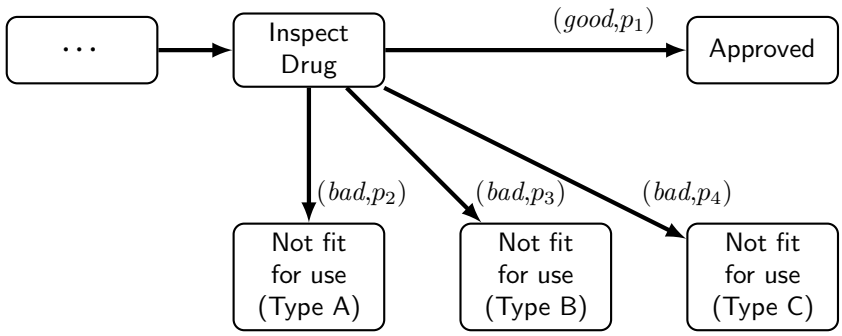


Figure 3.3: *Illustration of intention preserving stochastic branching.*

The behaviour depicted in Figure 3.3 essentially exhibits the stochastic semantics of a Markov Decision Process (MDP) [282], which provide an established mathematical framework for modelling this type of behaviour, which makes such

behaviour in a business process suited to formal analysis. Flows in a process graph will be assigned intentional preserving stochastic semantics by the following definitions:

Definition 3.6 (Flow Probability Function)
Given a process graph $P = (\mathbf{N}, \boldsymbol{\mathcal{F}})$, a decision point probability function is a partial function $\mathcal{P} : \mathcal{F} \times \mathbf{L} \rightarrow [0, 1]$ which for a given node $n \in \mathbf{N}$ and label $l \in \mathbf{L}$ assigns probabilities to all outgoing sequence flows $f = (n, x) \in \boldsymbol{\mathcal{F}}$ such that:

$$\sum_{\forall x \in \mathsf{out}(n)} \mathcal{P}((n, x), l) = 1$$

In practice, when the drug is deemed not fit for use the probabilistic choice of subsequent steps is based on observations of the frequency of various categories of fault. Well established methods in business process management allow for the determination of these frequencies (e.g. time-motion studies [270]).

3.2.3 Modelling Resources

When constructing models of business processes, the capacity to include numerical data greatly expands what can be deduced from a model. These additions are commonly known as *rewards* (or *costs*) [282], terms first introduced in connection with the development of Markov Models. In the business process domain the term *resources* is typically used to capture cases when a limited number of resources (people, systems, machines, etc.) are available to execute a process. These annotations must be relatively uncomplicated so that a business analyst can add them to a model in the graphical paradigm which dominates the modelling of business processes. However, most graphical paradigms neglect the inclusion of data in models [23], [207], [208] in a fashion that allows effective quantitative analysis [278], and instead focus only on their qualitative behaviour.

The term *reward* is somewhat misleading because there is no practical distinction between costs and rewards, and these annotated values can be used to keep track of whichever quantities may be of interest in a process. In this thesis, *rewards* (or *costs*) are added by associating real values with certain states or transitions of the process graph. This addition of reward annotations allows for reasoning not just about the qualitative behaviour of a business process, but also about a wider range of quantitative system measures.

A key benefit of including quantitative data in models, is to allow for the determination of bounds on the performance properties of such systems [142], which facilitates the early identification and exclusion of inefficient designs. There

is substantial evidence [108], [283] that being able to catch flaws and determine the performance properties of business processes early in their design phase can result in smoother integration, precise provisioning to meet service level agreements, and significant cost savings. For example, it is possible to compute properties such as *mean time to job completion*, *expected power consumption*, or *worst-case mean time to recovery from any failure state*. Further, by parametrising certain data elements associated with actions in a business process, it becomes possible, by means of quantitative formal analysis techniques, to automatically evaluate these properties for a range of values of a given parameter and determine system behaviour when limited resources are expended.

Definition 3.7 captures the notion that certain nodes have some reward or cost associated with being in the corresponding state in a business process task.

Definition 3.7 (Node Reward Function)
Given a node $n \in \mathbf{N}$ *within a process graph* $P = (\mathbf{N}, \boldsymbol{\mathcal{F}})$*, the node reward function is a partial function* $\mathcal{R}_{\mathcal{N}} : \mathbf{N} \to \mathbb{R}_{\geq 0}$.

It should be noted that Definition 3.7 defines a partial function, allowing the convenience of not defining rewards for all nodes. However, a total function can be obtained if one considers the reward function, $\mathcal{R}$ as mapping nodes for which node rewards are undefined to 0. An example of such a node reward annotated to a process graph is the annotation $[R_{problem}]$ seen in Figure 3.1 which records that an additional problem has occurred.

Definition 3.8 captures the notion that a reward is accumulated when transitioning between states.

Definition 3.8 (Flow Reward Function)
Given a flow n $(n_1, n_2) \in \boldsymbol{\mathcal{F}}$ *within a process graph* $P = (\mathbf{N}, \boldsymbol{\mathcal{F}})$*, the flow reward function is a partial function* $\mathcal{R}_{\mathcal{F}} : \mathbf{N} \times \mathbf{N} \to \mathbb{R}_{\geq 0}$.

An example of such a node reward annotated to a process graph is the annotation $[R_{treated}]$ seen in Figure 3.1, where it is used to record the treatment of a patient.

As many reward structures as desired can be associated with a given process graph, so that a single node or flow may have multiple separate numerical properties which are incremented when they are encountered. The flexibility to associate rewards with either nodes or flows makes modelling conceptually simpler, and allows for capturing unique cases which is not possible using node rewards.

In particular, flow rewards can be combined with probabilistic choice as defined in Section 3.2.2 to approximate the probabilistic accumulation of rewards according to an arbitrary probability distribution. For a discrete probabilistic reward distribution, this can be done by constructing flows $\mathbf{f}$ sharing a fixed label l for every sample in the distribution of the reward, and associating each flow $f \in \mathbf{f}$ with the corresponding probability and value of the reward for that sample by means of $\mathcal{P}_{f,l}$ and $\mathcal{R}_{\mathcal{F}}$.

This idea can be extended to continuous distributions by discretizing the distribution probability density function by taking means of *binning* samples that correspond to intervals from the distribution of reward values. Hence, given interval n of reward distribution D with bounds l_n and u_n the probability and associated reward value of an approximating flow f_n is given as:

$$p_n = Pr_D[l_n < X < u_n]$$

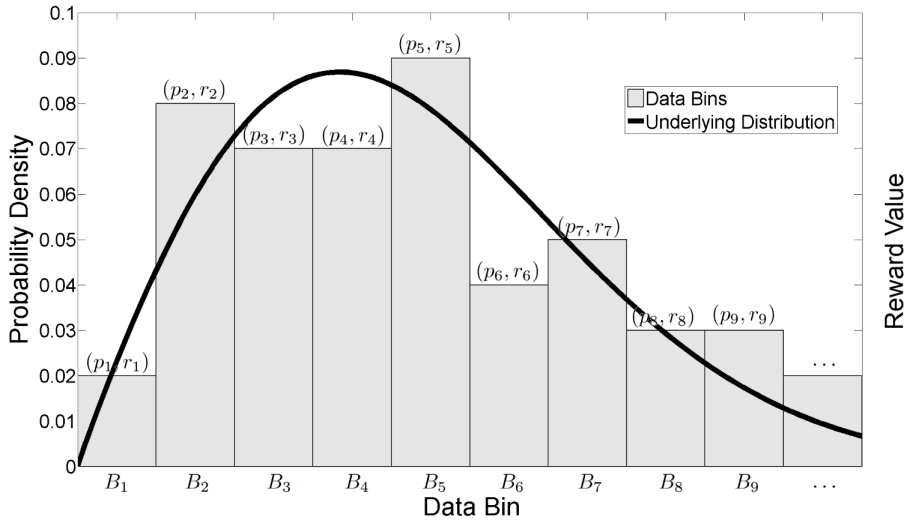
and

$$r_n = \frac{1}{2}(u_n - l_n).$$

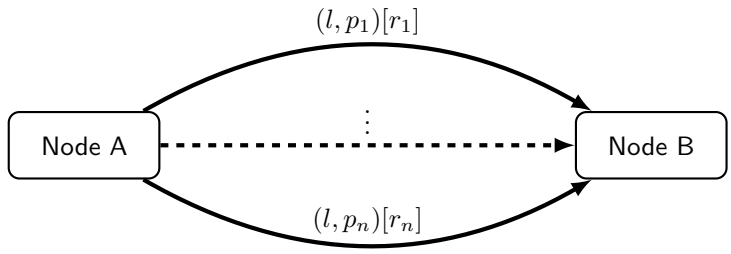
A useful technique for the binning of an arbitrary continuous distribution has been developed by Shimazaki and Shinomoto [262]. This technique determines the optimal bin width, for a chosen number of bins, so as to minimise the error between the constructed histogram and the underlying distribution. The error is estimated from the data itself such that if the underlying distribution is not known, as might be the case when a model is being constructed based on data observed from experiment, this method can still be applied.

Application of this approach to allow for the construction of reward flows which approximate a continuous distribution is illustrated in Figure 3.4. Here a reward structure is constructed which models the time between two steps in a process where the time taken is known to be given by a Weibull distribution [216]. A number of bins n, corresponding the number of flows to be used to approximate the distribution, is chosen and a set of samples of the distribution are determined. The method of Shimazaki and Shinomoto [262] is then employed to determine the optimal bin width as shown in Figure 3.4(a). The probabilities p_n and reward values r_n for each flow f_n can thus be determined as illustrated in Figure 3.4(b).

In Section 5.3.2 a method for automatically adjusting the number of bins used depending on the required accuracy of quantitative verification property is presented.



(a) Observed distribution



(b) Resultant transitions

Figure 3.4: *Illustration of the approximation of a stochastic continuously distributed reward by means of binning. l is a common label shared by all transitions.*

The parametrisation of reward structures is a further possible model annotation. Here, a reward annotation $[r_n]$ is appended with an associated upper bound u_n such that the full annotation appears as $[r_n : u_n]$. The semantic effect of this is discussed when dealing with the analysis of models in Section 5.3.1.

Finally, it is feasible that the restriction that the reward must be a positive real number and that it will be *monotonically increasing* during system execution can be loosened. A discussion of this is provided in Chapter 10, although no real-world models encountered have required this feature.

3.3 Business Process Model and Notation

Business process modelling as a discipline has traditionally suffered from a proliferation of process definition languages based on similar, but subtly different, concepts and constructs. After numerous attempts, standardization efforts have converged towards the BPMN language [207], which is intended for modelling business processes primarily during the analysis and design phases. It has emerged as a standard notation for capturing business processes, especially at the level of domain analysis and high-level systems design [71].

This section will extend the formalism of process graphs to capture the syntax and structural semantics of a subset of the BPMN language. The motivation for constructing a formalised variant of the BPMN language is based on the discussion in Section 2.4 and the following considerations:

1. BPMN has been established as a standard by the OMG and a considerable number of tutorials and guidebooks are available to assist practitioners in constructing models. Fundamentally, BPMN is a graphical notation for specifying business processes, with the stated primary goal of providing a notation that is readily understandable by all business users. BPMN's ability to serve as a standardized bridge between business process design and implementation has lead to widespread adoption, and it is now considered a rising industry standard [71].
2. The industrial partner in this PhD project has chosen to employ BPMN to model a number of workflows in the healthcare domain of the type described in Section 1.1.
3. BPMN has seen widespread adoption in the healthcare industry [237], [249], where the language design goals have made it relatively easy for hospital management to describe and communicate hospital workflows. Specific case studies [234], [235], [248] have underlined this by developing models of various complex hospital workflows rapidly and allowing manual restructuring of process flows.
4. Statistical observations [193] suggest that a core set of elements are overwhelmingly the ones that are actually employed in real-world business process modelling.

As described in the discussion of BPMN in Section 2.3.2.2, the language is a graphical notation only, and while the BPMN specification [207] provides extensive syntactic rules, the execution semantics of BPMN are "described informally (textually)" [207, page 425], leaving open to interpretation a number of issues such as the treatment of deadlocks and race conditions. The loosely

defined execution semantics of BPMN have motivated the development of a range of formalised semantics mostly with the ultimate aim of allowing some sort of automated formal analysis.

However, as noted in Section 2.3.2.2, none of the identified approaches to the formalisation of BPMN [88], [173], [200], [215], [230], [289], [295] are suited for use in the quantitative analysis of systems which have been extended to accommodate stochastic elements. The formalism of process graphs allows for the creation of an extension to construct a formal model of the core subset of BPMN through the inclusion of a set of structural semantic rules.

3.3.1 Core BPMN

The current version of BPMN, version 2.0, permits models to consist of a wide range of nearly 100 graphical elements, covering the description of many categories of tasks, events, errors, areas of responsibility, and general annotations. A large study of real-world BPMN usage [193], [238] found that the frequency of BPMN construct usage follows a broadly exponential distribution, as shown in Figure 3.5. The most used subset, which the authors labelled the *core* subset of BPMN, consists of only eight elements, and indeed more than 70% of models surveyed consisted only of these elements.

In spite of this vast range of graphical objects, there are essentially only two fundamental categories of object in BPMN, *nodes* and *flows*. It is therefore natural to extend process graphs (Definition 3.1) to capture the notation of BPMN models. Given a process graph $P_1 = (\mathbf{N}, \boldsymbol{\mathcal{F}})$, the various possible nodes of Core BPMN models can be mapped to subsets of $\mathbf{N}$ that cover the different categories of nodes that form Core BPMN. The sequence flows of a Core BPMN model can be mapped directly to flow relations $\boldsymbol{\mathcal{F}}$ of a process graph. Likewise, message passing between pools in a Core BPMN model can be captured by the synchronisation arcs for process graphs defined in Definition 3.5. Finally Core BPMN makes use of *Pools*, a structural element which helps organise nodes within BPMN model by grouping nodes into sets and can be captured by functions assigning membership of nodes to a set.

The elements of Core BPMN are shown in Figure 3.6.

Tasks shown in Figure 3.6(a) are subsets of the nodes of a process graph, and are the basic actions done as part of a given business process, e.g. “sending a letter” or “administering a drug”. The set of all tasks of a core BPMN model is denoted by $\mathbf{T}$.

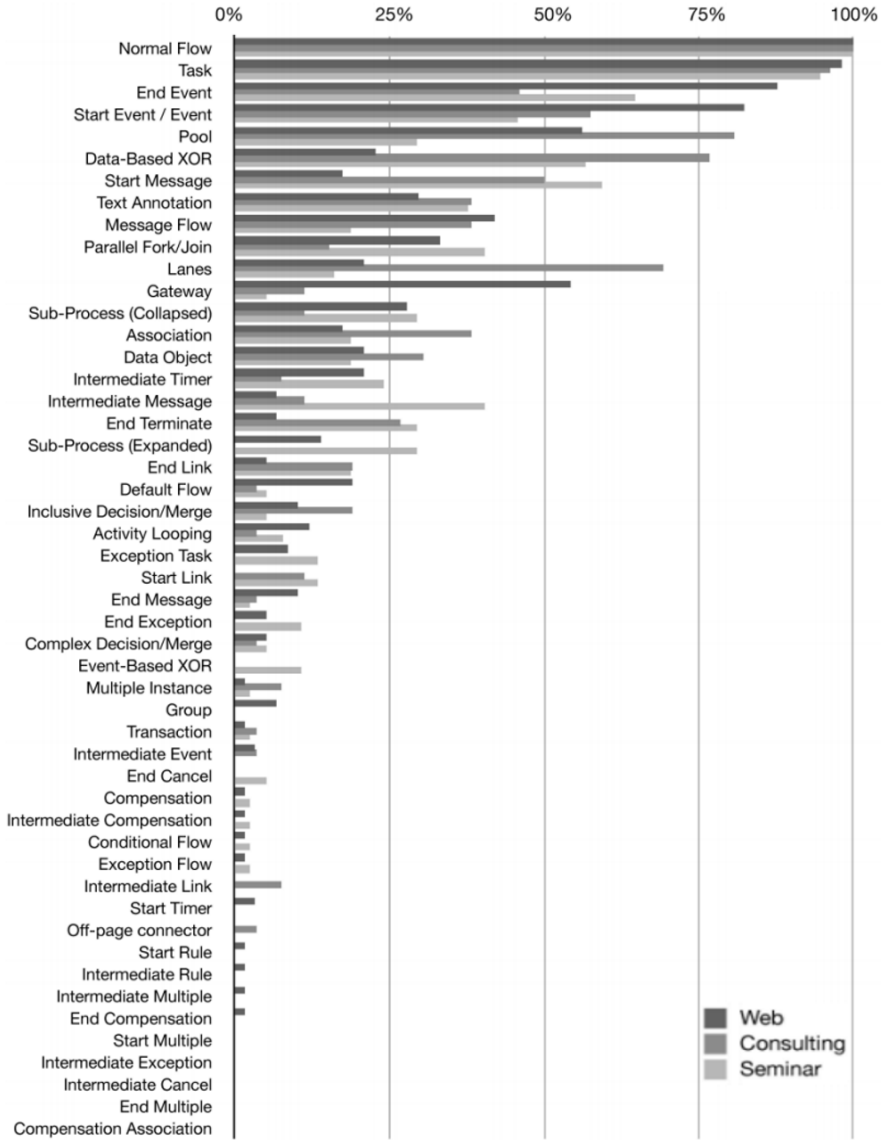


Figure 3.5: *Real world usage of BPMN constructs (source [193]).*

Start/End Events shown in Figure 3.6(b) and Figure 3.6(c) respectively, are subsets of the nodes of a process graph and are the points at which a business process starts or stops. These can be considered as a special type of task which have the property that they have respectively no incoming or

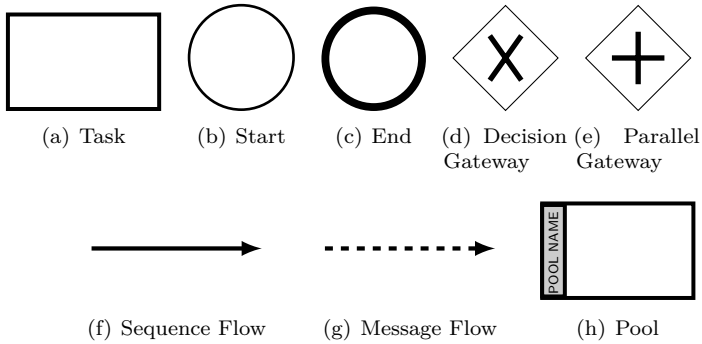


Figure 3.6: *Core BPMN elements.*

outgoing sequence flows. The sets of start and end events are denoted by $\mathbf{E} \subseteq \mathbf{E^S}$ and $\mathbf{E^E}$, where the disjoint sets $\mathbf{E^S}$ and $\mathbf{E^E}$ respectively represent start and end events. In terms of modelling business processes, these are organisational elements in BPMN and do not reflect actual steps of a business process.

Exclusive Decision Gateways shown in Figure 3.6(d) are also subsets of the nodes of a process graph, and are elements which model decision points in a business process at which one of a number of possible actions are chosen, as illustrated in Figure 3.7(c). According to the BPMN specification: "a decision can be thought of as a question that is asked at a particular point in the process. The question has a defined set of alternative answers. Each answer is associated with a *conditional expression* that is associated with a Gateway's outgoing Sequence Flows." [207, page 290]. The set of exclusive decision gateways of a core BPMN model is denoted by $\mathbf{G^D}$.

Parallel Gateways shown in Figure 3.6(e), are subsets of the nodes of a process graph and are used to create (fork) or combine (merge) parallel flows. These can take the possible forms shown in Figure 3.7(a) and Figure 3.7(b) respectively, i.e. either merging a number of sequence flows or splitting a single sequence flow into multiple flows. Specifically the BPMN specification states that: "a parallel gateway creates parallel paths without checking any conditions; each outgoing sequence flow receives a token upon execution of this gateway. For incoming flows, the Parallel Gateway will wait for all incoming flows before triggering the flow through its outgoing sequence flows." [207, page 294]. The set of all gateways of a core BPMN model is denoted by $\mathbf{G} = \mathbf{G^M} \cup \mathbf{G^F} \cup \mathbf{G^D}$ where $\mathbf{G^M}$, $\mathbf{G^F}$ and $\mathbf{G^D}$ are disjoint sets of merge-, fork- and the previously defined decision- gateways.

Sequence Flows shown in Figure 3.6(f), are subsets of the flows of a process graph, and are used to model the sequence in which tasks are performed. Visually, a sequence flow links one BPMN element to another. The set of all sequence flows of a core BPMN model is denoted by $\mathcal{S}$.

Message Flows shown in Figure 3.6(g), are synchronisation arcs between process graphs, and are employed to pass messages between separate processes. This is achieved by means of synchronisation in the fashion described in Section 3.2.1. The set of all message flows of a core BPMN model is denoted by $\mathcal{M}$.

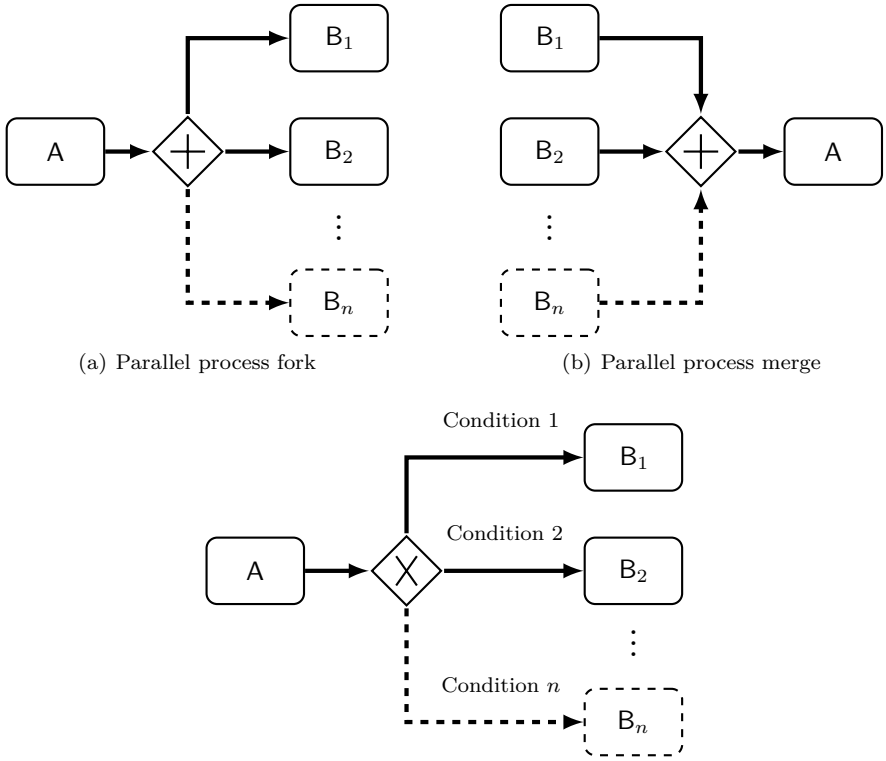
Pools shown in Figure 3.6(h), are organisational elements which denote that what is contained within such an element is part of a single set of activities performed by a chosen entity. These are not directly mapped to process graphs but can be expressed as functions which partition the set of nodes of a process graph. The set of pools is denoted by $\mathbf{P}$.

The process of modelling a workflow in BPMN involves composing a number of BPMN elements into a Business Process Diagram (BPD). The intention is that a business process diagram captures the complete workflow of a business process, with separate sub-components of a workflow organised into separate pools. A BPD is formally defined here as a simple partition of a process graph with synchronisation arcs as defined in Definition 3.1, with the straightforward addition of a function to capture pools.

Definition 3.9 (Core BPD)
A core BPD is an extended process graph tuple $BPD = (\mathbf{N}, \boldsymbol{\mathcal{F}}, \mathbf{P}, \mathsf{pool}, \mathbf{L}, \mathsf{lab})$ *where* $\mathbf{N} \subseteq \mathbf{T} \cup \mathbf{E} \cup \mathbf{G}$*, is a set of nodes composed of the following disjoint sets:*

- *Tasks* $\mathbf{T}$*, are the basic actions performed as part of a business process.*
- *Events* $\mathbf{E} \subseteq \mathbf{E^S} \cup \mathbf{E^E}$*, where the disjoint sets* $\mathbf{E^S}$ *and* $\mathbf{E^E}$ *respectively represent start and end events.*
- *Gateways* $\mathbf{G} \subseteq \mathbf{G^D} \cup \mathbf{G^F} \cup \mathbf{G^M}$*, where the disjoint sets* $\mathbf{G^D}$*,* $\mathbf{G^F}$ *and* $\mathbf{G^M}$ *respectively represent exclusive decision gateways, parallel fork gateways, and parallel merge gateways.*

$\boldsymbol{\mathcal{F}} \subseteq \mathcal{S} \cup \mathcal{M}$ *is a set of flow relations, where sequence flows* $\mathcal{S} \subseteq \mathbf{N} \times \mathbf{N}$ *relate nodes to each other and message passing* $\mathcal{M} \subseteq \mathbf{T} \times \mathbf{G^M}$ *is a relation between tasks and parallel merge gateways.* $\mathbf{P} \subset \mathfrak{P}(\mathbf{N})$ *is a set of disjoint pools and* $\mathsf{pool} : \mathbf{N} \to \mathbf{P}$ *maps nodes to a pool* $p \in \mathbf{P}$*.* $\mathbf{L}$ *is a set of unique labels and* $\mathsf{lab} : \boldsymbol{\mathcal{F}} \to \mathbf{L}$ *is a labelling function which assigns labels to flows.*



(a) Parallel process fork

(b) Parallel process merge

(c) Exclusive decision gateways

Figure 3.7: *General forms of gateway (branch) constructs in core BPMN.*

3.3.2 Structural Semantic Rules for Core BPMN Models

Because BPMN lacks a formal semantics and since a key objective of this thesis is to perform formal analysis of BPMN models, a set of structural semantic rules (well-formedness conditions) are imposed so as to impose the minimum semantic interpretation necessary to determine the control flow of a BPD. In most cases, no more semantic interpretation is added than explicitly given in the standard [207]. The details of the execution semantics of BPMN models are explored in Chapter 5. It should be noted that the approach taken here to imposing structural semantics is similar to the approach of Wong and Gibbons [289] and Ouyang et al. [215].

The definition of a BPD given in Definition 3.9 models business processes by using elements of $\mathcal{S}$ to define a directed graph with nodes which are elements of $\mathbf{N}$, with elements of $\mathcal{M}$ used to connect separate process graphs. However,

Definition 3.9 allows for graphs which are unconnected, do not have start or end elements, and are free-form or have various other properties which deviate from the BPMN standard.

However, the structural semantics of BPMN is not entirely undefined and the standard does provide some constraints. To ensure that a BPD complies with the BPMN standard [207] and describes a meaningful business process, a set of structural semantic rules are imposed which enforce restrictions on connecting elements, pool boundaries, and message passing. These are intended to impose no more semantic interpretation than is implied by the standard, and are only required to be able to determine the control flow of a model. These conditions are defined using the functions from Definitions 3.2 to 3.4, and the functions defined in Definition 3.10 and Definition 3.11 :

Definition 3.10 (Pool Contents)
Given a BPD, the nodes contained within a given pool $p \in \mathbf{P}$ *are given by the function:*

$$\mathsf{contents}(p) = \{n \in \mathbf{N} | \mathsf{pool}(n) = p\}$$

Definition 3.11 (Path Contents)
Given a BPD, the nodes contained within a given path between nodes a *and* b *are given by the path contents function:*

$$\mathsf{path}(a, b) = \{n \in \mathbf{N} | a\mathcal{S}^* n \wedge n\mathcal{S}^* b\}$$

where $\mathcal{S}^*$ *is the reflexive transitive closure of* $\mathcal{S}$

Using these definitions, the following conditions are imposed on a *structurally sound* BPD and are discussed below:

Definition 3.12 (Structurally Sound Core BPD)
A BPD is structurally sound if the following conditions hold:

$$
\begin{array}{ll}
\textbf{\textit{E1}} & \forall e \in \mathbf{E^S} : \mathsf{in}(e) = \emptyset \wedge |\mathsf{out}(e)| = 1 \\
\textbf{\textit{E2}} & \forall e \in \mathbf{E^E} : |\mathsf{in}(e)| = 1 \wedge \mathsf{out}(e) = \emptyset \\
\textbf{\textit{T1}} & \forall t \in \mathbf{T} : |\mathsf{out}(t)| = 1 \\
\textbf{\textit{G1}} & \forall g \in \mathbf{G^D} : |\mathsf{in}(g)| = 1 \wedge |\mathsf{out}(g)| \geq 2 \\
\textbf{\textit{G2}} & \forall g \in \mathbf{G^D} : \forall t \in \mathsf{out}(g) : \mathsf{lab}(t) \neq \bot \\
\textbf{\textit{G3}} & \forall g \in \mathbf{G^F} : |\mathsf{in}(g)| = 1 \wedge |\mathsf{out}(g)| \geq 2 \\
\textbf{\textit{G4}} & \forall g \in \mathbf{G^M} : |\mathsf{in}(g)| \geq 2 \wedge |\mathsf{out}(g)| = 1 \\
\textbf{\textit{G5}} & \forall (s,e) \in \mathbf{E^S} \times \mathbf{E^E} : \\
 & \quad |\{g \in \mathbf{G^F} | g \in \mathsf{path}(s,e)\}| = |\{g \in \mathbf{G^M} | g \in \mathsf{path}(s,e)\}| \\
\textbf{\textit{P1}} & \forall n \in \mathbf{N} : \exists p \in \mathbf{P} : \mathsf{pool}(n) = p \\
\textbf{\textit{P2}} & \forall n \in \mathbf{N} : \mathsf{pool}(n) = p_1 \wedge \mathsf{pool}(n) = p_2 \implies p_1 = p_2 \\
\textbf{\textit{P3}} & \forall p \in \mathbf{P} : |\mathsf{contents}(p) \cap \mathbf{E^S}| = 1 \\
\textbf{\textit{P4}} & \forall p \in \mathbf{P} : \forall n \in \mathsf{contents}(p) : \exists (s,e) \in \mathbf{E^S} \times \mathbf{E^E} : s\mathcal{S}^* n \wedge n\mathcal{S}^* e \\
\textbf{\textit{S1}} & \forall (n_1,n_2) \in \mathbf{N} \times \mathbf{N} : n_1 \mathcal{S} n_2 \implies \mathsf{pool}(n_1) = \mathsf{pool}(n_2) \\
\textbf{\textit{M1}} & \forall (n_1,n_2) \in \mathbf{N} \times \mathbf{N} : n_1 \mathcal{M} n_2 \implies \mathsf{pool}(n_1) \neq \mathsf{pool}(n_2) \\
\textbf{\textit{M2}} & \forall (n_1,n_2) \in \mathbf{N} \times \mathbf{N} : n_1 \mathcal{M} n_2 \implies (n_1 \in \mathbf{T}) \wedge (n_2 \in \mathbf{G^M})
\end{array}
$$

where $\mathcal{S}^*$ is the reflexive transitive closure of $\mathcal{S}$.

E1 and **E2** impose the restriction on start and end events that they respectively do not have in or out flows, and that they are followed or respectively preceded by a single node. These restrictions are in line with the notational requirements [207, pages 238, 246] and the stated semantics of BPMN [207, pages 439-444].

T1 ensures that tasks do not branch process sequence flow, although the semantics of BPMN imply that tasks may in fact branch process flow. This mechanism is described in terms of implicit gateways [207, page 427]. **T1** therefore effectively expresses a restriction on BPMN models that ensures that branching must be made explicit by means of gateways, hence this restriction does not limit the expressiveness of structurally sound core BPDs relative to the full BPMN semantics, but simply makes it slightly more verbose. While implicit gateways can be made automatically explicit by means of a preprocessing step when analysing models; this is in line with the 7PMG guideline **7PMG-G4** of ensuring that models are well structured to force a modeller to make the behaviour explicit.

In structurally sound core *BPD*s, exclusive decision gateways take the general form shown in Figure 3.7(c) and the choice of a resultant path depends on an external condition. **G1** ensures that exclusive decision gateways only have a single inflow and that they must have multiple outflows. This behaviour is a

restriction of the full BPMN semantics which allows for multiple inflows [207, page 435]. The motivation for this restriction is to simplify the analysis performed in Chapter 5. However, multiple inflow gateways can be simulated in structurally sound core *BPD*s by using a dummy task prior to the gateway to gather incoming flows. **G2** ensures that the outflows of a decision gateway are labelled, which is implied by the BPMN standard.

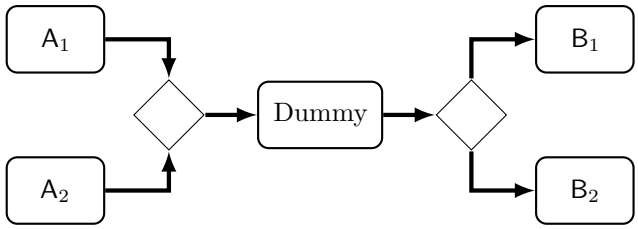
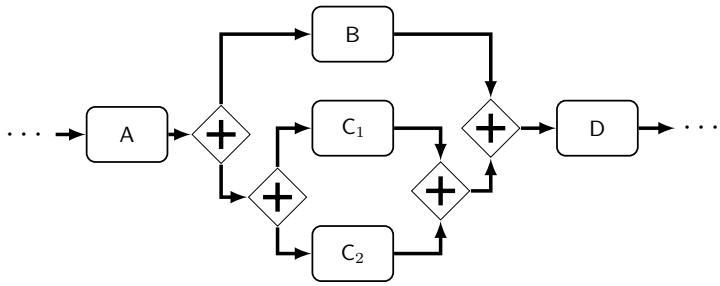


Figure 3.8: *Simulation of a gateway (decision or parallelising) which allows an arbitrary number of in- and out- flows, using multiple structurally sound core BPMN gateways and a dummy state.*

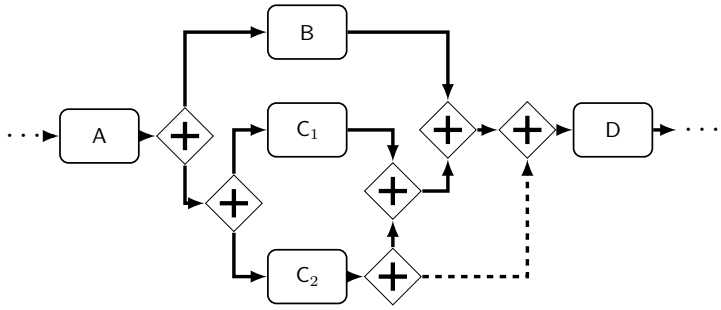
Parallel fork and merge gateways have, in core *BPD*s, the general forms shown in Figures 3.7(a) and 3.7(b) respectively. **G3** ensures that parallel fork gateways have a single inflow and multiple outflows, while conversely **G4** ensures parallel merge gateways only have a single outflow and multiple inflows. These rules are restrictions of the full BPMN standard [207, page 434] which allow a single parallel gateway to have multiple in- and out- flows. The restrictions are again made to simplify the analysis developed in Chapter 5; however, it should be noted that they can be avoided by adding a dummy task between merging and forking gateways to simulate gateways which have both multiple in- and out-flows, as illustrated by Figure 3.8.

G5 expresses the requirement that parallel blocks are *properly nested*. This means that an inclusive gateway only merges sequence flows originated by another inclusive forking gateway and that there is an one-to-one correspondence between the forking and the merging inclusive gateways. Further this implies that the cardinality of the set $\mathbf{G^F} \cup \mathbf{G^M}$ is an even number. The concept of proper nesting is illustrated in fig. 3.9, where examples of both properly and improperly nested gateways are shown. However, requirement **G5** is not explicitly made in the BPMN standard, and as such this rule is a further restriction of the standard. This requirement has also been imposed by Aalst [17] who coined the term *well-structuredness* in context of employing *workflow nets* [18] to model BPMN processes. Here well-structuredness denotes that "A model is well-structured if the split/join constructions are properly nested" and is a requirement to allow

their formal analysis. In addition, the work of Kiepuszewski et. al. [155] suggests methods to redesign improperly nested models to arrive at well-structured equivalents for a large number of cases.



(a) Properly Nested



(b) Improperly Nested

Figure 3.9: *Illustration of* proper nesting*, model (a) features properly nested parallel control flow blocks, whereas model (b) features a jump out of the inner control structure (dashed elements) thus violating structural semantic rule* ***G5****.*

P1 ensures that all nodes are contained within a pool, and **P2** ensures that no node can be a member of more than one pool. **P3** ensures that each pool only contains one start event, and **P4** ensures that all nodes within a given pool lie on a path of sequence flows from a start to an end event. All of these conditions are strictly in line with the BPMN standard [207, pages 22-25].

In addition, **S1** and **M1** ensure that messages passed between nodes always cross a pool boundary and that sequence flows do not cross pool boundaries, matching the BPMN standard [207, page 120].

The definition of a *BPD* given here for Core BPMN requires messages to be passed from a task to a parallel merge gateway. This is not in line with the BPMN standard, but the implied semantics and examples in the standard suggest that a message must be received before further progression of the receiving process is possible. For this reason, messages are required to be linked to parallel merge gateways to make this behaviour explicit.

As the most commonly used core fragment of BPMN does not include the *interaction node* [193], the constraint **M2** ensures that messages are passed from tasks, within a pool, to parallel merge gateways, in a different pool. This is not in line with the BPMN standard, but the implied semantics and examples in the standard suggest that a message must be received before further progression of the receiving process is possible. For this reason, messages are required to be linked to parallel merge gateways to make this behaviour explicit. Note that the construction of Core BPMN as an extension of *process graphs* allows the asynchronous synchronization mechanism developed in Section 3.2.1 to be employed directly.

It should be noted that the approach to message passing in the BPMN standard makes use of the separate mechanisms of either a *message catch event*, a direct connection to a task or a message which can be passed to a pool as a whole. The motivation for introducing the simplified message passing used in Core BPMN in this thesis is that it simplifies the subsequent analysis and reduces the number of constructs used to model processes, making this modelling formalism easier to use for practitioners. Further, note that the definition of a *BPD* given here does not differentiate between *initiating* and *non-initiating* messages, as required in the full BPMN standard. Again, this choice is made to keep the language simple and allow messages to provide a formal synchronization mechanism as defined in Definition 3.5.

A further structural semantic rule imposed on business processes is that sequence flows may not intersect.

Definition 3.13 (Prevention of non-deterministic parallelism)
A BPD is a structurally sound stochastic Core BPD if parallelism is not used to implement choice, i.e. there are no intersecting sequence flows in the BPD.

This restriction prevents cases such as the one illustrated in Figure 3.10 where the flows between parallel merging and forking gateways connect to separate nested blocks, allowing infinite recursion. While the BPMN standard does not appear to forbid this construct, this behaviour is not required for modelling real world business processes and a number of other approaches of formal analysis of

BPMN do require such a restriction. Of particular note is the work of Aalst [17] on workflow nets where crossover points, termed *handles* in the context of Petri-nets, are disallowed due to the authors opinion that parallelism should not be employed to implement choices. Further it should be noted that a goal for BPMN [207, page 1] is the possibility for conversion into executable Business Process Execution Language (BPEL) processes. However, BPEL [211] models have the restriction that they are only well-structured if points of crossover are not present. Finally work by Gruhn and Laue [121] on developing complexity metrics for business processes suggests a considerable increase in model complexity when cross-over points are allowed and a considerable number of business process models which exhibit points of cross-over contained errors. Hence this restriction helps ensure that Core BPMN models are in line with the 7PMG guideline **7PMG-G4** of ensuring that models are well structured.

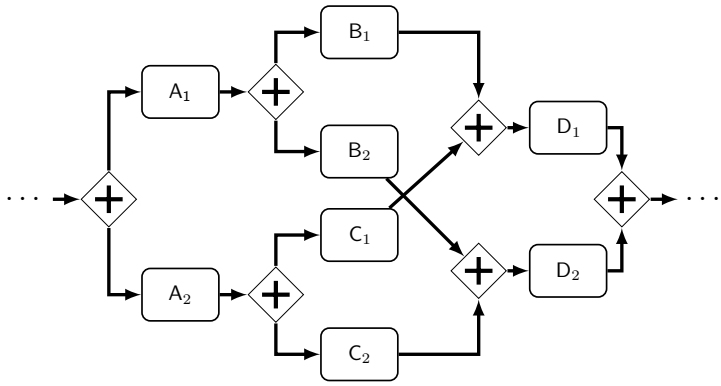


Figure 3.10: *Illustration of the crossover problem for nested parallel Core BPMN processes, which leads to infinite recursion. Note that this model can not be restructured so as to avoid the transitions from* B_2 *and* C_1 *intersecting.*

Overall, the negative effects of the restrictions placed upon Core BPMN in comparison with the full BPMN language are very limited. They primarily serve the purpose of making models more explicit, through restrictions on the branching of sequence flows as well as limiting in- and out- flows of gateways. Compared to the full BPMN language these restrictions only have the effect of making models more circuitous and any full BPMN language process that only makes use of the constructs of Core BPMN can be converted into a Core BPMN process through the modifications presented in this section. With respect to the restrictions on the cross-over of sequence flows and the requirement for proper nesting Core BPMN does limit the range of possible models that can be constructed. However, this is in line with various modelling guidelines and is common in most formal approaches to business process analysis. While it seems

to be the case that such models frequently are constructed in error [121], [122] a limited number do describe meaningful business processes and it should be stressed that these specific cases cannot be modelled in Core BPMN. However, it has not been the case working with our industrial partner that any business workflows were encountered which could not be modelled with Core BPMN.

3.3.3 Extending Core BPMN

This section presents some extensions of the Core BPMN language made possible by the construction of BPDs as restrictions upon process graphs. These allow for the modelling of systems with features that go beyond what is possible with traditional BPMN. These extensions are strict extensions to Core BPMN. As such, no alteration of the existing Core BPMN elements defined in Section 3.3.1, or the structural semantics of core BPMN defined in Section 3.3.2, or even the full BPMN language, is required to add these features, only annotations to a BPD are required.

To enable stochastic branching, the extension to process graphs described in Section 3.2.2 is adopted. BPMN already makes use of external conditions on decision gateways to select the outgoing flow from a decision point. The outcome of these decisions is modelled by the set $\mathbf{L}$, and assigned to specific flows by the function lab introduced in Definition 3.2. This extension is explicitly part of the BPD and not the process graph. This means that Definition 3.6 can be instantiated for BPDs as a restricted version of the original definition:

Definition 3.14 (BPD Gateway Flow Probability Function)
Given a BPD, a decision gateway probability function is a partial function $\mathcal{P} : \mathcal{S} \times \mathbf{L} \to [0, 1]$ which for a node $g \in \mathbf{G^D}$ and label $l \in \mathbf{L}$, assigns probabilities to all outgoing sequence flows (g, x), such that for a given l:

$$\sum_{\forall x \in \mathsf{out}(g)} \mathcal{P}((g, x), l) = 1$$

The structural soundness condition **G2** ensures that all outflows of a decision gateway are labelled. Note that different outflows may have the same label. Definition 3.14 ensures that all decision gateways have an associated probability and that the sum of all probabilities for a given label l is 1. Figure 3.11 illustrates the application of $\mathcal{P}$ to a decision gateway g.

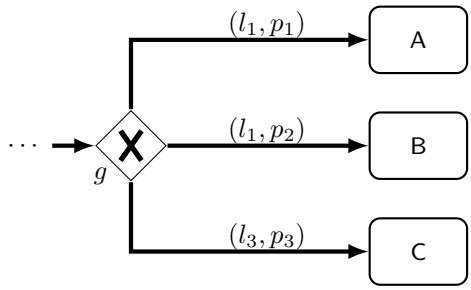


Figure 3.11: *Assignment of label probability pairs to a decision gateway. Here application of $\mathcal{P}$ requires $p_1 + p_2 = 1$ and $p_3 = 1$.*

Note that the function defined above is a partial function and some sequence flows may not have associated probabilities. Moreover, for the purposes of this framework, probabilities are only allowed to be attached to decision gateways, as these points are the only meaningful places to speak about probabilistic transitions. Hence, in this framework when a decision gateway occurs there must be an associated probability, even if it is 1.0. This is done by introducing two additional structural soundness conditions:

Definition 3.15 (Structurally Sound Stochastic Core BPD)
A BPD is a structurally sound stochastic Core BPD if it is structurally sound and the following additional conditions hold:

X1 $\forall g \in \mathbf{G^D} : \forall n \in \mathsf{out}(g) : \mathcal{P}(g \mathcal{S} n)$ *is defined.*
X2 $\forall m \in \mathbf{N} \setminus \mathbf{G^D} : \forall n \in \mathbf{N} : \mathcal{P}(m \mathcal{S} n)$ *is undefined.*

It should be noted that in a practical implementation of this modelling approach it is convenient to omit probabilities of 1.0 on flows when modelling and have these added automatically as a preprocessing step before analysis.

A key benefit of including quantitative data in models is to allow for the determination of bounds on the performance properties of such systems [142], as this facilitates the early identification and exclusion of inefficient designs. To enable reward annotations for Core *BPD*s, Definition 3.7 and Definition 3.8 are employed, directly as defined for process graphs, to add the needed annotations. Additionally, parametrised reward structures can be added simply by including a bound in the same fashion as for process graphs. As many rewards as desired can be associated with a given *BPD*, so that a single node or flow may have multiple different numerical properties which are incremented when they are encountered.

The flexibility to associate rewards with either nodes or flows makes modelling conceptually simpler, and, as shown in Section 3.2.3, allows for capturing unique cases which could not be captured with the other type of reward.

3.3.4 Excluded constructs

Because of the very basic nature of the elements of Core BPMN, it is believed [56] that most of the individual elements of the entire BPMN language for private (non-communicating) processes can be simulated by combining several Core BPMN elements. However, there are some elements of the full BPMN language which can not be directly captured by means of process graphs. The most noteworthy of these will be discussed below.

The full BPMN language allows for abstraction of business processes where elements of a model are undefined. An example of this in the BPMN standard is the notion of *public* and *private* processes shown in Figure 3.12. Here a fully defined public process (Doctor) is shown interacting with an abstracted process (Patient) where only the pool containing the process is defined. Clearly processes for which an explicit definition does not exist can not be reasoned about formally and therefore this capability has been excluded from the formalised Core BPMN language.

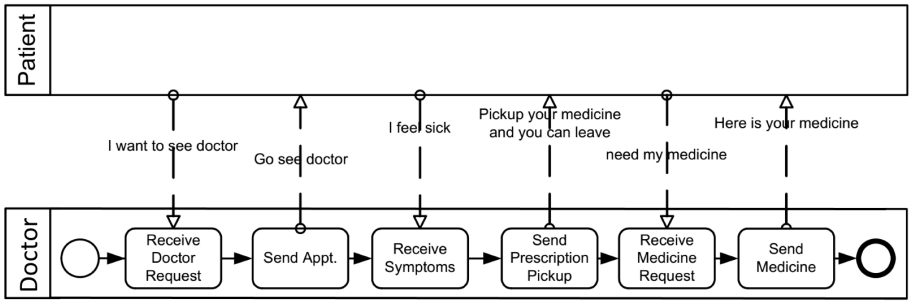
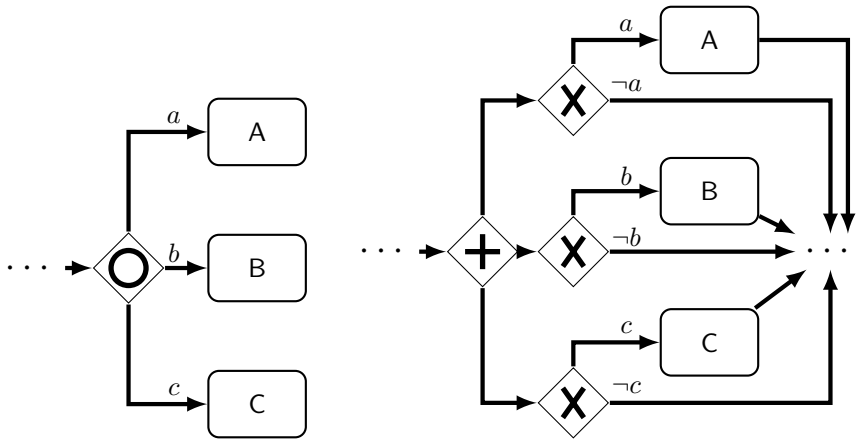


Figure 3.12: *Example of an abstracted BPMN process, source [207, page 24].*

Furthermore, a number of annotational elements have been excluded from the description of Core BPMN. These elements, such as *groups* [207, page 68] or *text annotations* [207, page 71], can be included in Core BPMN models. However, these elements add no semantic detail to a BPMN model and as such no further treatment is made of these elements when models are analysed.

Of particular note, in terms of excluded constructs, are two types of gateways which pose unique problems when attempting formal analysis. The first of these is the *complex gateway* [207, page 295] which is a control flow element which allows for arbitrary logic to be executed based on the inputs from one or more inflows and producing outputs on one or more outflows. Formal analysis of this construct is extremely challenging due to the possibility for arbitrary logic to be executed in determining control flows. Similar to the problem with the abstraction of processes, any approach to formalising this gateway would in turn require the formalisation of any possible process embedded in the gateway.

Secondly, the *inclusive gateway* [207, page 292] is perhaps the key construct which is excluded from this formalisation of the BPMN language. This gateway exist in two forms, a *diverging* and *converging* form. The diverging form allows all outgoing flows for which a condition is satisfied to be invoked in parallel, distinguishing it from the exclusive decision gateway where it is required that only one of the outgoing flows is chosen. The motivation for excluding the *diverging* version of the inclusive gateway is that it can be readily simulated by means of a combination of parallel forks and exclusive decision gateways, as illustrated in fig. 3.13. Therefore the inclusion of this construct would needlessly bloat the number of constructs in Stochastic Core BPMN, violating recommendation **7PMG-G1** of the 7PMG guidelines [186].



(a) Divergent inclusive gateway (b) Equivalent Core BPMN process

Figure 3.13: *Simulation of a divergent inclusive gateway (a) by means of Core BPMN constructs (b).*

Convergent inclusive gateways, also known as inclusive OR joins, have been studied in a number of contexts including Event-driven Process Chain (EPC)s [122], BPMN [72], [92], [100] and Yet Another Workflow Language (YAWL) [292]. Inclusion of these gateways present a more complex problem in that the specification of their semantics includes a non-trivial and non-local backwards search of the flow graph of the BPD [72], [100], due to the requirement that all tokens that can be "expected" to arrive at the gateway must arrive before execution can continue.

However, work by Favre and Völzer [100] has characterised the possibilities for replacing convergent inclusive gateways with combinations of exclusive decision and parallel gateways. They identify three categories of convergent inclusive gateway usage which may be either locally replaced, non-locally replaced or which can not be replaced at all. For both the local- and non-local- replacement cases they also provide polynomial time algorithms for both the identification and replacement of convergent inclusive gateways. For the final class of irreplaceable convergent inclusive gateways they provide a proof of the impossibility of their replacement. It should be noted that these replacement techniques do not alter the soundness of the process, i.e., they cannot introduce or fix a control-flow error.

It should be noted that the semantics of BPMN 2.0 have changed to prevent use of convergent inclusive gateways leading to deadlock in the case of the so called *vicious circle* example. First identified by Kindler et al. in the context of EPCs [27], this is a class of situations in which two inclusive gateways depend on each other cyclically. However, in spite of these changes the vicious circle example may still exhibit race-conditions in BPMN 2.0 [72].

The excluded constructs described in this section do impose a limit on the range of models which can be constructed in Core BPMN relative to the full BPMN language. However, only a subset of possible uses of convergent inclusive gateways prevent employing a preprocessing step to replace non-Core BPMN elements. Further, it should be noted that 7PMG Guideline **7PMG-G5** suggests that process modellers seek to avoid the inclusive OR-join. While no data exists specifically on OR-join usage in BPMN models, in the case of Event Driven Process Chains, 67% of models in a large survey [122] did not make use of OR-joins at all. Of the remainder that did use inclusive OR-joins, 62% of joins could be locally replaced. For the remaining set of 57 OR-joins 45 could be non-locally replaced and a further 10 were on closer inspection found to be faulty models. In fact in this study only 2 models, out of an original set of 285 models, remained where an OR-join was necessary. This raises the question of to what extent there is real-world demand for convergent inclusive gateways (OR-joins). Crucially the core BPMN set presented in this thesis has been found adequate to model the type of processes described in Section 1.1.

3.4 BPMN Modelling Example

To demonstrate a complete structurally sound (well-formed) core BPMN model extended with stochastic non-determinism and rewards, a simplified example of a business process describing the medical assessment and treatment of a patient is used. This business process is shown as a BPD in Figure 3.14, and is an example of the *collaboration pattern* of BPMN models.

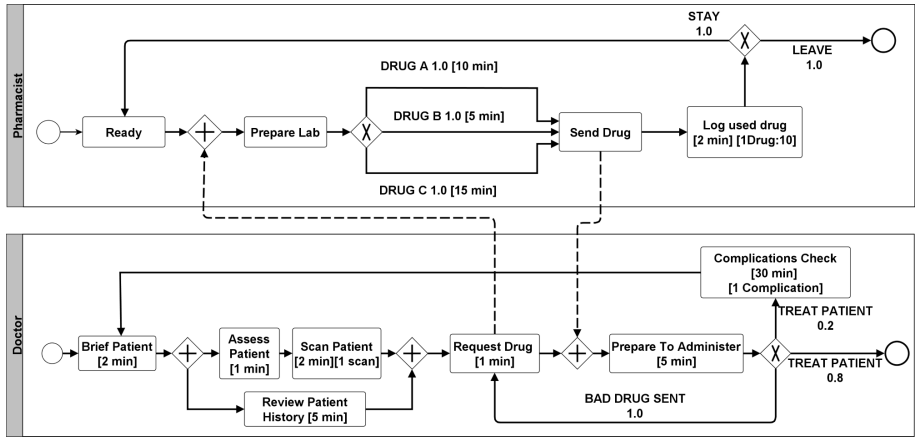


Figure 3.14: *Core BPMN model of a collaborative medical workflow of a doctor ordering drugs from a pharmacy, extended with stochastic non-determinism and rewards.*

In this example, the BPD contains two pools representing the roles of a medical doctor and a pharmacist supplying drugs to the doctor respectively. The BPD is a structurally sound stochastic Core BPD with reward elements.

The doctor begins his workflow from the start event of the doctor pool, by briefing a patient. This task is annotated with a *reward* which records 2 minutes of time spent on this task on each visit to this state. After this stage, the sequence flow indicates that a parallel fork is made and two sets of tasks are performed simultaneously: the assessment of the patient and subsequent scanning. While this is happening, the doctor reads the patient's medical history. The sequence flow is merged again once both sets of tasks are completed (parallel sub-flows). Following this, the doctor always chooses, in this simple example, to request a drug to treat the patient and passes a message to the pharmacist. The actions are annotated with time rewards and a separate reward structure which records the number of times a medical scanning device has been used.

Meanwhile, the pharmacist process has progressed from its start event to preparing the lab, and is now waiting for a message from the doctor requesting a drug. When this message is received, the pharmacist makes a non-deterministic, but not probabilistic decision, to prepare one of three drugs: A, B, or C based on the message received from the doctor. In each case, the transition representing preparation of the drug is annotated with a reward expressing the time taken to prepare that drug. Having prepared the chosen drug, he then sends a new message to the doctor which represents passing the drug to the doctor. Having dispatched the drug, the pharmacist then logs that the drug has been used, an action which has multiple reward structures which capture the time taken and the use of a unit of stored drugs. In this case the combined storage of drugs is limited to 10 units (regardless of the choice of type of drug), and this is indicated in the associated annotation.

The drugs sent by the pharmacist allow the doctor process to progress, and the doctor next prepares the drugs for administration to the patient. During the preparation of the drug the doctor checks if the drugs are fit for use, and at this stage a non-deterministic decision, depending on a factor outside the control of the business process, is made by the doctor to either treat the patient or resolve that a *bad* dose of drugs has been sent. Note that at this point two steps take place as dictated by the gateway's flow probability function; a choice is made by the doctor between treating the patient or declaring that the prepared drug is bad and not fit for use as dictated by the labels of the outflows of the gateway. Following this choice there are various probabilistic outcomes of the decision made. In the case of a bad dose, there is a probability of 1.0 that the doctor process loops back to the earlier task where he requests a new dose. In the case when the doctor chooses to treat the patient, the application of drugs has a probability of 0.8 of being successful, and if it is not, with 0.2 probability the patient incurred a complication which is also modelled by a reward. When not successful, and after a 30 minute wait (reward) for adverse reactions, the process loops back to a previous state and, in this example, the doctor eventually tries administering the drug again.

After preparing each drug the pharmacist makes an exclusive choice between staying at work or going home, and in the case when the pharmacist stays at work, the process loops back to a state where the pharmacist is ready to receive another drug order.

3.5 Denotational semantics of Core BPMN

This section provides an *denotational semantics* for Core BPMN as a formalization of the meaning of business processes described by means of Core BPMN models. A *traces model* is used to define the meaning of a process graph as the set of sequences of events (*traces*) that the process can be *observed* to perform.

This approach is inspired by the model of *trace semantics* developed for Communicating Sequential Processes (CSP) by Tony Hoare in [137], [138]. In this model a process is taken to denote a set of communication traces, built from events which represent abstract records of communication. A trace here represents a partial history of the communication sequence occurring when a process interacts with its environment; since communication is synchronized an input or output event really stands for a potential for communication. And since traces record a partial behaviour it is natural to work with (non-empty) prefix-closed sets of traces.

A *Core BPMN trace* as defined in Definition 3.16 describes one specific possible sequence of execution of a BPMN process. The trace is a sequential record of all the events occurring during the course of the execution of the business process up to some moment in time, i.e. all nodes traversed during the execution of a BPD. A trace of the behaviour of a process is a finite sequence of symbols recording.

The trace takes the form of a sequence of tasks and syntactic Core BPMN events that the process can be observed to perform. All events are recorded and can be considered as being recorded by a perfect observer who sees all events. If the observer sees multiple events happening simultaneously, he or she is permitted to write down the events in an arbitrary order.

When constructing a trace based model of the execution of Core BPMN models, the notion of *sub-processes* is convenient. Here *sub-processes* are *partitions* of a larger BPD as illustrated in Figures 3.16 and 3.17, which highlight the selection of two sub-processes from a BPD. Sub-processes are partitions of a larger BPD and therefore will not conform to the *structural* semantic constraints defined in Definition 3.12. Hence sub-processes are not in themselves BPDs, but will exhibit the same control flow behaviour required of a BPD.

The set of traces for a specific BPD is constructed compositionally, by the recursive application of the traces function to each element of a BPD as dictated by the sequence- and message- flow relations $\mathcal{S}$ and $\mathcal{M}$. In Definition 3.16 the recursive application of traces to *unwind* a BPD can succinctly be constructed by means of a lambda expression [73].

Definition 3.16 (Process Traces)
Given a core BPMN BPD $BPD = (\mathbf{N}, \boldsymbol{\mathcal{F}})$ and $A \subset BPD$ a subprocees of the BPD, the function $\mathsf{traces}(BPD) \subseteq \Sigma^*$ *is defined as:*

$$\mathsf{traces}(x\mathcal{S}A) = \{\mathsf{LFP}(\lambda A.x \frown \mathsf{traces}(A))\}$$
$$\mathsf{traces}(x\mathcal{M}A) = \{\mathsf{LFP}(\lambda A.x \frown \mathsf{traces}(A))\}$$

where Σ^* is the set of all possible finite sequences of elements of $\mathbf{N}$, LFP denotes a least fixed point *computation, and $\frown$ denotes the concatenation of sequences.*

The specific elements of a BPD are addressed as follows:

The generation of traces for the simple case of sequential execution of tasks is shown in Figure 3.15. In this case a trace simply consists of the sequence of tasks executed. Note that this process contains *start* and *end* events which are treated in the same fashion as *tasks*.

Figure 3.15: *Illustration of trace construction for a BPMN BPDs featuring sequential tasks.*

Formally the following rules are applied to generate the traces for *start*, *end* and *task* elements prefixed a sub-process A:

1. **Start elements**: $s \in \mathbf{E^s} : \mathsf{traces}(s\mathcal{S}A) = \{< s > \frown x | x \in \mathsf{traces}(A)\}$
2. **End elements**: $e \in \mathbf{E^e} : \mathsf{traces}(e) = \{< e >\}$
3. **Task elements**: $t \in \mathbf{T} : \mathsf{traces}(t\mathcal{S}A) = \{< t > \frown x | x \in \mathsf{traces}(A)\}$

Applying rules 1, 2 and 3 to the entire model shown, X, in Figure 3.15 thus yields the trace

$$\mathsf{traces}(X) = \{< s, t_1 > \frown x \frown < t_2, e > | x \in \mathsf{traces}(\cdots)\}$$

The case of generating traces for BPMN models that feature non-determinism in the form of exclusive decision gateways is handled by generating the trace for each possible sequence of events from a exclusive decision gateway. Each possible execution path is denoted as an ordered sequence of events and multiple traces are generated at each exclusive decision gateway corresponding to the different execution paths from the gateway. Formally this trace construction is given by the following rule:

4. **Exclusive Decision Gateways** (denoted $\otimes$) for sub-processes A_1 to A_n connected as sequence flows from an exclusive decision gateway g_d produce the following traces:

$$\mathsf{traces}\big(\otimes_{g_d}(A_1,\ldots,A_n)\big) = \bigcup_{i=1}^{n}\Big\{ < g_d > \frown x | x \in \mathsf{traces}(A_i)\Big\}$$

An example of the general form of the employment of an exclusive decision gateway is shown in Figure 3.16. In this figure a decision gateway allows for a choice between two possible paths.

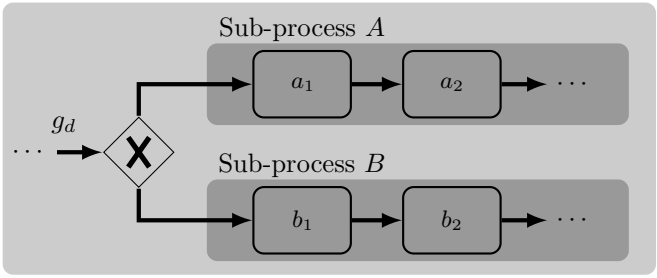


Figure 3.16: *Illustration of trace construction for BPMN BPDs featuring non-determinism (exclusive decision gateway).*

In Figure 3.16 the traces for the entire model shown, X, when the trace construction rule 4 is applied are:

$$\begin{aligned}\mathsf{traces}(X) =& \{x \frown < g_d > \frown y | x \in \mathsf{traces}(\cdots) \wedge y \in \mathsf{traces}(A)\} \\ & \cup \{x \frown < g_d > \frown y | x \in \mathsf{traces}(\cdots) \wedge y \in \mathsf{traces}(B)\}\end{aligned}$$

Addressing the case of generating traces for cases of *concurrency* and *recursion* requires the introduction of an *interleaving* operator. Formally the interleaving of *sub-processes* is defined as:

Definition 3.17 (Interleaving)
Given a core BPMN BPD and with chosen sub-processes $A_1, \ldots, A_n$ *with traces* $\mathsf{traces}(A_i) =< a_{i,1}, a_{i,2}, \ldots >$. *The interleaving function* Inter *produces all possible elements of* $\mathsf{traces}(A_i)$ *for all* i *that respect the individual partial ordering of* $\mathsf{traces}(A_i)$.

Definition 3.17 produces traces from both sub-processes that are arbitrarily interleaved in time. For example the interleaving of two sub-processes X and Y for which $\mathsf{traces}(X) =< x_1, x_2 >$ and $\mathsf{traces}(Y) =< y_1, y_2 >$ produce the following set of traces:

$$\mathsf{Inter}\big(\mathsf{traces}(X), \mathsf{traces}(Y)\big) = \left\{ \begin{array}{l} < x_1, x_2, y_1, y_2 >, < x_1, y_1, x_2, y_2 >, < x_1, y_1, y_2, x_2 >, \\ < y_1, x_1, x_2, y_2 >, < y_1, x_1, y_2, x_2 >, < y_1, y_2, x_1, x_2 > \end{array} \right\}$$

In the BPMN [207] standard the operational semantics for the parallel execution of processes is largely undefined. The approach taken in this thesis with regard to concurrency allows execution to produce all possible interleavings of concurrent events. The motivation for this choice is driven by Objective 2a where safety properties of a business model are of interest. By considering all possible interleavings the maximum model with regard to possible executions is explored, ensuring the detection of all possible safety violations.

The case of generating traces for a BPMN BPD featuring concurrent *(parallel)* behaviour involves the interleaving of the traces of all sub-processes executed in parallel. Formally this is given by the following rule:

5. **Parallel regions** (denoted⫼) encompassing separate sub-processes $A_1, \ldots, A_n$, between a pair of fork and merge gateways g_f and g_m produce the following traces

$$\mathsf{traces}\left(\underset{(g_f, g_m)}{⫼} (A_1, \ldots, A_n) \right) = \bigcup_{x \in \mathsf{Inter}\left(\mathsf{traces}(A_1), \ldots, \mathsf{traces}(A_n)\right)} \{< g_f > \frown x \frown < g_m >\}$$

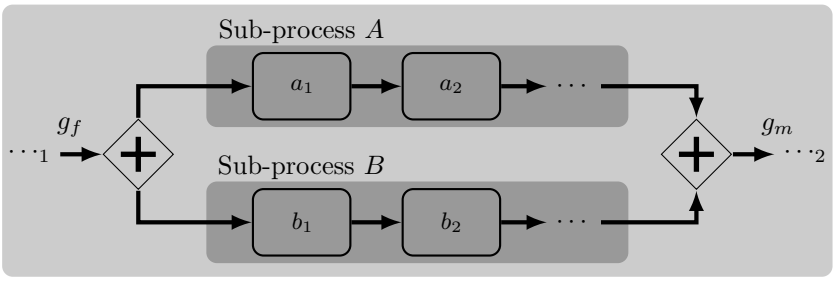


Figure 3.17: *Illustration of trace construction for BPMN BPDs featuring concurrency (parallel fork and merge gateways).*

Applying rule 5 to the entire model, X, shown in Figure 3.17 yields the following traces:

$$\mathsf{traces}(X) = \bigcup_{x \in \mathsf{Inter}\left(\mathsf{traces}(A), \mathsf{traces}(B)\right)} \left\{ \begin{array}{l} p \frown< g_f >\frown x \frown< g_m >\frown q \mid \\ p \in \mathsf{traces}(\cdots_1) \wedge q \in \mathsf{traces}(\cdots_2) \end{array} \right\}$$

As defined in Definition 3.16 producing traces for BPMN BPDs incorporating recursion requires the recursive application of the traces. Hence, recursion in BPMN models produces traces of infinite length as the tasks which are performed recursively will be repeated. This is formally expressed using the following rule:

6. **Recursion** For a process X containing a sub-process A, which is invoked recursively, the following traces are produced:

$$\mathsf{traces}(X) = \bigcup_{n \in \mathbb{N}} \left\{ \left[x \frown< g_d > \mid x \in \mathsf{traces}(A) \right]^n \right\}$$

   Where $[\cdot]^n$ denotes n iterations of concatenation.

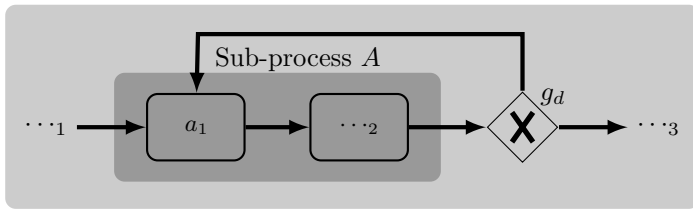


Figure 3.18: *Illustration of trace construction for a BPMN model featuring recursion.*

In the example model shown, X, in Figure 3.18 when applying rule 6 the following traces are produced:

$$\mathsf{traces}(X) = \bigcup_{x \in \mathbb{N}} \left\{ \begin{array}{c} p \frown (< a_1 > \frown q \frown < g_d >)^n \frown r \mid \\ p \in \mathsf{traces}(\cdots_1) \wedge q \in \mathsf{traces}(\cdots_2) \wedge r \in \mathsf{traces}(\cdots_3) \end{array} \right\}$$

The least fixed point computation LFP of Definition 3.16 admits infinite chains and therefore may not terminate. These infinite traces reflect the possibility of an arbitrary number of iterations of the recursive region of a BPD. In the analysis of Core BPMN models driven by Objective 2a where safety properties of a business model are of interest, will include an arbitrarily large, but finite, number of iterations before the recursion region is exited. Hence allowing the maximal behaviour of recursive behaviour to be explored with respect to safety properties.

The synchronisation between BPMN models is catered for in the BPMN standard [207, pages 43,93,120]. However, the description of synchronisation (message passing) requires a number of additional syntactic elements solely employed to describe synchronisation. In addition standard BPMN does not provide an unambiguous description of the execution semantics of message passing. Therefore, the synchronisation (message passing) mechanism of process graphs described in Section 3.2.1 is employed as the semantic interpretation imposed during analysis. In this asynchronous setting, input and output are not treated symmetrically: output actions can occur without requiring the participation of another process. However, a process wishing to perform an input action must wait, if necessary, until the relevant message is emitted from an external process (pool).

An illustration of the execution semantics of message passing for Core BPMN models is shown in Figure 3.19. In this figure a task in sub-process X passes a message to a merge gateway in sub-process Y and may later receive another message from sub-process Y. Transitions out of sub-sub-process B_1 require the reception of a message before proceeding. However the progression of sub-process X from a_x to sub-sub-process A_2 has no execution restrictions.

Formally, message passing (denoted $\rightarrow$) is addressed by the following rule:

7. **Message flows** where sub-process $X = \cdots, a_n, \cdots, a_x, \cdots, a_m, \cdots$ transmits a message to sub-process $Y = \cdots, b_n, \cdots, b_x, \cdots, b_m, \cdots$ in sub-process B. Here we define sub-sub-processes: $A_1 = \cdots, a_n, \ldots$, $A_2 =$

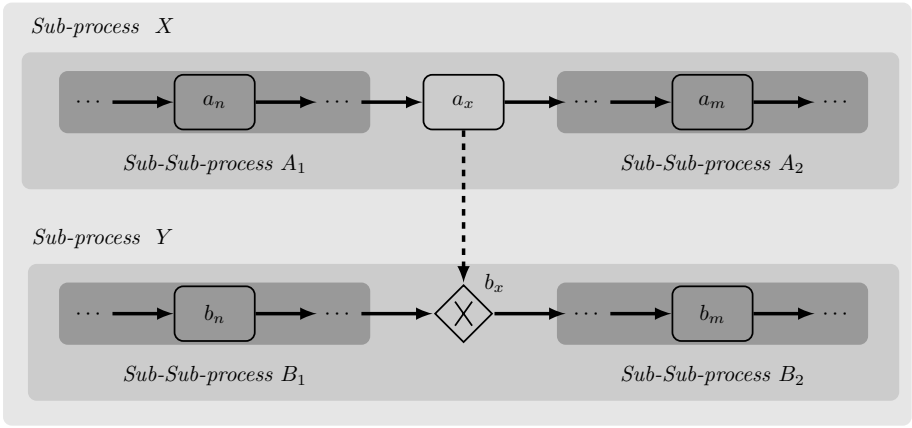


Figure 3.19: *Trace decomposition for message passing between pools A and B.*

$\ldots, a_m, \ldots$ $B_1 = \ldots, b_n, \ldots$ and $B_2 = \ldots, b_m, \ldots$.

$$\mathsf{traces}(X \to Y) = \left\{ \begin{array}{c} x \frown < a_x > \frown y \ \mid \\ x \in \mathsf{Inter}\big(\mathsf{traces}(A_1), \mathsf{traces}(B_1)\big) \\ \wedge y \ \in \mathsf{Inter}\big(\mathsf{traces}(A_2), \{< b_x > \frown t | t \in \mathsf{traces}(B_2)\}\big) \end{array} \right\}$$

All traces of Core BPMN BPDs have the following properties given in Definition 3.18:

Definition 3.18 (Trace Properties)
Given a core BPMN BPD $BPD = (\mathbf{N}, \boldsymbol{\mathcal{F}})$*, the set of traces of* $\mathsf{traces}(BPD)$ *has the following properties:*

1. $\langle\rangle \in \mathsf{traces}\,(BPD)$
2. $s_1 \frown s_2 \in \mathsf{traces}\,(BPD) \implies s_1 \in \mathsf{traces}\,(BPD)$

Definition 3.18 implies that the set of traces for a given Core BPMN BPD BPD is non-empty as it always contains the empty trace $\langle\rangle$. In addition, if a concatenated sequence $s_1 \frown s_2$ is an element of $\mathsf{traces}(BPD)$ then s_1 is an element of $\mathsf{traces}(BPD)$, i.e. $\mathsf{traces}(BPD)$ is prefix-closed.

Two BPDs P_1 and P_2 are considered *trace equivalent* if

$$\mathsf{traces}(P_1) \cong \mathsf{traces}(P_2)$$

that is to say to an external observer both processes may be observed to execute the same tasks in the same order. Note that two BPDs being trace equivalent does not imply that P_1 and P_2 are identical, but that to an external observer they will be seen to have the same behaviour.

Core BPMN is a stochastic model of business process execution. Therefore each trace, of *finite length*, has a particular chance of occurring. The probability $\mathbb{P}$ of an element of a trace, given the probability P of an individual state, can be compositionally constructed using the following rules:

1. $\mathbb{P}(<>) = 1$
2. $\mathbb{P}(< a > \frown T) = P(a) \cdot \mathbb{P}(T)$

Likewise the expectancy of trace $\mathbb{E}$, given the expectancy E of an individual state, can be compositionally constructed using the following rules:

1. $\mathbb{E}(<>) = 1$
2. $\mathbb{E}(< a > \frown T) = E(a) \cdot \mathbb{E}(T)$

Note that infinite sequences of events are possible in a Core BPMN BPD due to recursion. In these cases the probability and expectancy of any specific trace is 0.

Note that during the analysis of BPMN BPDs an imposed semantics given in Section 4.5.2 is applied where all possible traces are explored as a single statespace.

3.6 Chapter Summary

This chapter defined the formalism of *process graphs* and defined a notation for synchronization between process graphs. Process graphs were extended to include stochastic behaviour and rewards. A denotational semantics was defined for process graphs based on ideas from CSP. Placing restrictions upon process graphs, by means of a set of structural semantic rules, allowed for the formalisation of the most commonly used elements of the BPMN language. This formalisation allowed for BPMN to be extended with stochastic behaviour and rewards and hence amenable to quantitative analysis. Finally an denotational, trace based, semantics was developed for Core BPMN.

An example inspired by the motivating problem from Section 1.1 was presented in this extended formalised variant of BPMN and was annotated with stochastic branching and a variety of rewards to illustrate the developments of this chapter.

Chapter 4

Formal Methods

"We argue that proof construction is unnecessary in the case of finite state concurrent systems and can be replaced by a model-theoretic approach which will mechanically determine if the system meets a specification expressed in propositional temporal logic. The global state graph of the concurrent systems can be viewed as a finite Kripke structure and an efficient algorithm can be given to determine whether a structure is a model of a particular formula (i.e. to determine if the program meets its specification)." (Edmund M. Clarke 1981)

Overview

This chapter provides a brief overview of the main formal methods approaches in informatics. These approaches are surveyed and their key strengths and weakness identified. This leads to the choice of model checking as the approach to employ for the analysis of the type of stochastic business processes considered in this thesis. The main current quantitative model checking software tools are reviewed. Discussion of their key capabilities motivates the choice of the Probabilistic Symbolic Model Checker (PRISM) model checker to perform this analysis. A formal description of the PRISM modelling and query languages are given, along with a description of the semantics imposed on a model when analysed by PRISM.

A formal foundational description of quantitative stochastic model checking is given in Appendix A. A number of issues described in relation to formal verification were presented in [9].

4.1 Formal Methods

The proliferation of complex systems in all aspects of our lives places an increasing importance on the need for them to function correctly. The conventional method of checking that a system behaves as intended, testing it on a representative set of scenarios, is often inadequate in the face of the increasing complexity of these systems. Today, the main serious approaches to addressing the challenge of constructing and verifying properties of systems are commonly known as *formal methods*, which is simply the application of mathematics, specifically formal logic, to the design and analysis of systems.

In general, the application of formal methods aims to provide a mathematical proof of properties of interest on an abstract mathematical model of the system. The correspondence between the formal model and the nature of the system, at least for the properties being proven, is intended to be guaranteed by construction. However, assumptions about the system's environment are hard to formulate explicitly, and there will always be cases when a system is deployed in an environment for which it was not originally designed. As such formal methods will never provide a route to make systems 100% secure, but they can be seen as the best known approach to verify the properties of system models.

The following subsections provide a brief description of each of the main categories of formal methods approaches to the verification of systems. This motivates the choice of *model checking* as the formal method to employ to analyse and verify Stochastic Core Business Process Model and Notation (BPMN) models of business processes in line with objectives 2a, 2b and 2c.

4.1.1 Theorem proving

Automated theorem proving is concerned with the mechanization of formal reasoning following the laws of logic. In essence this approach seeks to emulates human mathematicians by constructing a proof, given some underlying logic, by means of various deductive rules.

The roots of this approach go back to the end of the previous century when Frege developed his *Begriffsschrift* [1] the first comprehensive effort to develop a formal language suitable as a foundation for mathematics. However, Bertrand Russell discovered a paradox which showed that Frege's system was inconsistent and hence the truth of any proposition can be derived in it. To remedy this Russell then devised his own system based on a type theory and he and Whitehead demonstrated in their seminal work *Principia Mathematica* how this could serve as a formal foundation of mathematics. Later, Hilbert developed a simpler alternative, the *predicate calculus*. From these developments first-order logic emerged as an analysis of the most fundamental basis for the notion of mathematical proof and rose to become the dominant mathematical paradigm of the 20th century, the *first-order system*. This system emerged as the logic that is necessary and sufficient for codifying mathematical proofs, axiomatizing mathematical theories, and studying their metatheory. [102]

These developments were reformulated and extended by Gentzen in the first half of the twentieth century into a system of *natural deduction* where the meaning of each logical connective is explained via inference rules which was to be a seminal development in this field. While Gentzen was motivated by a desire to establish the consistency of number theory, he was unable to prove the needed *cut elimination theorem* directly for natural deduction. For this reason he introduced an alternative system, *the sequent calculus* and showed that it derives the same theorems as natural deduction. He proceeded to prove the cut elimination theorem for both classical and intuitionistic logic. This is widely regarded as the first consistency proof for a formal logical system [42], following this he proceeded to establish that all true propositions in the sequent calculus

[1] Literally translated as "concept notation" and presented as "a formula language, modelled on that of arithmetic, of pure thought".

could be proven according to a simple strategy i.e. that the system was complete. The sequent calculus is widely considered the first deductive system that was found to be sound and complete [102].

Fundamentally Automated theorem proving seeks to solve the problem automatically determining whether a set of axioms logically imply a hypothesis. Formally this can be expressed as follows:

Definition 4.1 (Theorem Proving Problem)
Given a set axioms $\mathbf{A} = a_1, \ldots, a_n$ *and a hypothesis* H *the model checking problem is to determine if:*

$$\mathbf{A} \models H$$

where the satisfaction relation $\models$ *is defined in a formal mathematical logic.*

This problem can be viewed as a search and the strategies employed by automated deduction systems are either directly based on or can be derived from Gentzen's sequent calculus. Automated deduction approaches can be broadly classified as either working backwards from a proposed theorem toward the axioms, or forward from the axioms toward the theorem. Among the backward searching procedures are tableaux, connection methods, matrix methods and some forms of resolution. The most common forward searching methods are classical resolution and the inverse method [132]. However, many modern theorem proving tools, such as *VAMPIRE* [245], seek to combine both forward and backward seeking approaches.

The prominence of *resolution* in both forward and backward approach is no accident, this was an approach introduced by John Alan Robinson in 1965 and represented a major leap forward in the state of the art of theorem proving. Resolution is a rule of inference leading to a refutation theorem-proving technique for sentences in propositional logic and first-order logic. In other words, iteratively applying the resolution rule in a suitable way allows for telling whether a propositional formula is satisfiable and for proving that a first-order formula is unsatisfiable; this method may prove the satisfiability of a first-order satisfiable formula, but not always, as it is the case for all methods for first-order logic. This limitation in constructing proofs for first-order logic, and other logics of equivalent or greater expressibility, can be explained by Gödel's incompleteness theorems [42] that establish inherent limitations of all but the most trivial axiomatic systems capable of performing arithmetic.

The first incompleteness theorem states that no consistent system of axioms whose theorems can be listed by an "effective procedure" is capable of proving all truths about the relations of the natural numbers (arithmetic). For any such

system, there will always be statements about the natural numbers that are true, but that are unprovable within the system. The second incompleteness theorem, an extension of the first, shows that such a system cannot demonstrate its own consistency. As such Gödel's incompleteness theorems provide a fundamental limit to what can be achieved in theorem proving. Tarski's undefinability theorem, stated and proved by Tarski in 1936, further limits what can be proven, by showing that truth in the standard model of a system cannot be defined within the system [42]. Additionally, the Church–Turing thesis [42] places further limitations on what can be achieved by automated theorem proving. In simple terms, the Church–Turing thesis states that a function is algorithmically computable if and only if it is computable by a Turing machine. Hence there may be situations where automated theorem proving may be attempting to establish results that are undecidable or which are beyond calculation by means of a Turing machine (computer).

In practical terms an ideal theorem prover would be able to verify properties such as proving for the program shown in fig. 4.1 that the boolean data type *vaild* never takes on the value of `true`. However, this would be equivalent to proving *Fermat's Last Theorem*, which is a highly complex mathematical proof [284] that was only constructed after more than three centuries of effort. Currently such proofs are completely beyond what can be achieved with automated (or assisted) theorem proving. As a proof has been constructed for this theorem we know that the limiting results of Gödel, Tarski and Church–Turing do not apply. However, here the limitation is simply the length of the proof and the infinite search space of possible deductions in number theory.

```
// Determines if Fermat's Last Theorem is true for a power p
boolean Fermat(int p){
  int a ← 1
  boolean valid = false
  while (¬valid)
    for (int b ← 1, b++, b ≤ a)
      for (int c ← 1, c++, c ≤ (a + b)){
        if (a^p + b^p = c^p) then
          valid = true
        a ← a + 1
      }
  return valid
}
```

Figure 4.1: *Example of a program which can not be verified for $p \geq 3$.*

When employing theorem proving a choice must be made as to the logic within which theorem proving will be attempted. For the historical reasons mention above and despite being only semi-decidable, first-order logic is by far the most mature sub-field of theorem proving. In addition, it should be noted that first order logic can encode any Turing-computable problem, while having a well-defined, widely understood and even intuitive semantics. Many other logics can be reasonably, in the sense that well developed methods exist for doing so, translated to first-order logic. Further, reasoning about it has been found to be highly automatable, in theorem proving terms. This is due to a number of sound and complete calculi for proof searches in existence [132] and the development of search procedures which are reasonably efficient. In general first order logic has come to dominate theorem proving as it strikes a powerful balance between expressiveness and automatability.

Despite the many years of development of automated theorem proving techniques the fundamental problem of theorem proving is that from a computational point of view the search performed by theorem proving in first order logic is done over an infinite search space. This means that searches cannot be performed naively by an exhaustive search but must make use of various heuristics to be able to efficiently compute a path to a proof. Fundamentally, the central problem of how to search for a proof, by coupling a chosen inference system with a search plan to form a theorem-proving strategy has not been resolved in a fashion that allows for efficient general purpose theorem proving [51].

Despite the development of a number of highly sophisticated automated theorem proving systems, the state of the art in the form of *Coq* [45], *Isabelle* [206] and *VAMPIRE* [245], which has won the world cup in first order theorem proving twenty seven times [163], are still quite limited in what they are able to prove, frequently requiring human assistance to produce even simple proofs. In particular in the domain of proving properties of stochastic systems theorem provers have had limited development and no significant result, as in proving previously unknown theorems, has been achieved. Indeed, most well-established automated theorem proving tools do not even feature axiomatisations of the necessary measure theory which underpins probability theory.

Automated theorem proving tools also have some fundamental limitations in terms of usability. Constructing the required models of both the implementation of - and the specification for - a system to be verified in formal first order logic makes employing this approach a major challenge. When constructing these models it is necessary to make explicit many detailed assumptions and handle all special cases explicitly. This is shown by the extensive development of the Thousands of Problems for Theorem Provers (TPTF) language [269] for problem specification and associated library of standard problems. This format, which is

now well-established as the standard input for automate theorem proving tools, is a highly explicit block structured language into which translating graph-based models, such as those of stochastic Core BPMN, is extremely challenging.

In addition, when employing automated theorem proving to verify properties of a system it should be noted that a full specification for the system is required and any change to the specification will require constructing an entirely new proof in which it may not be possible to reuse the previous proof. Finally, when theorem proving fails it is likely not to be the case that a counterexample can be found. Frequently theorem proving may halt when a counterexample has been found for a sub-goal, but this may merely imply that the theorem prover is pursuing a bad proof strategy. Finally, it should be noted that the proofs generated by an automated theorem prover due to the need to address all assumptions and special cases can be extremely verbose and the proof often lack an underlying structure, making even the atypical case of a short proof difficult to understand.

In summary, as a method for the analysis and verification of business process formally defined as described is Chapter 3, automated theorem proving has the following key properties:

Main Strengths:

- Possible to specify any property expressible in first order logic.
- Verification result is a formal mathematical proof.

Main weaknesses:

- Limiting results in logic (Gödel, Tarski and Church-Turing results imply that not even elementary number theory can be done completely automatically).
- Computationally expensive to verify properties (often requires human assistance).
- Difficult to specify properties for verification (Need to make many hidden assumptions explicit).
- A full specification must be formulated (adding verification properties may require entirely new proof strategies).
- Lack of support for verification of stochastic properties (a large number of additional axioms would be needed).
- Lack of counterexamples generated when a verification property does not hold.
- Mapping Core BPMN Business Process Diagrams (BPDs) to a provable model format (.e.g. TPTP) is difficult.

- Automatically generated proofs may be very large (and consequently difficult to understand).

4.1.2 Static Analysis

The technique of static analysis is a varied set of approaches to deriving a safe and computable approximation of the dynamically arising behaviour in an executable system. The key insight of static analysis is that the behaviours of the system can be abstracted into a decidable over- or under- approximation which preserves the specific properties of interest in an abstract version of the system. Specifically, static analysis techniques seek to determine properties that apply to *all possible execution paths* of the system by exploring all possible paths of execution in an abstract version of the system. More formally the static analysis problem can be defined as:

Definition 4.2 (Static Analysis Problem)
Given a model M with a semantics of execution, and ϕ a property of interest, the static analysis problem is to determine an abstraction A of M which preserves ϕ and determines if execution of A satisfies ϕ. This can be expressed as determining if

$$M \models_A \phi$$

where the satisfaction relation $\models_A$ is defined as the property ϕ being present in an abstraction of M.

The goal of static analysis is to derive a computable semantic interpretation at some point. For instance, one may choose to represent the state of a computer program manipulating integer variables by forgetting the actual values of the variables and only keeping their signs ($+$, $-$ or 0). For some elementary operations, such as multiplication, such an abstraction does not lose any precision: to get the sign of a product, it is sufficient to know the sign of the operands. For some other operations, the abstraction may lose precision: for instance, it is impossible to know the sign of a sum whose operands are respectively positive and negative.

This approach is fundamentally limited by *Rice's theorem* which states that, for any non-trivial property of partial functions, there is no general and effective method to decide whether an algorithm computes a partial function with that property [141]. Where non-trivial means there exists both a partial function that has a given property and one that does not. Static analysis seeks to sidestep the limitations imposed by Rice's theorem by providing approximate

answers to non-trivial questions about executable systems. This approximation is conservative, meaning that the answers only err to one side; analyses either determine if a property *may* (when over-approximation is employed) or *must* (when under-approximation is employed) be true for a given executable system as illustrated in Figure 4.2.

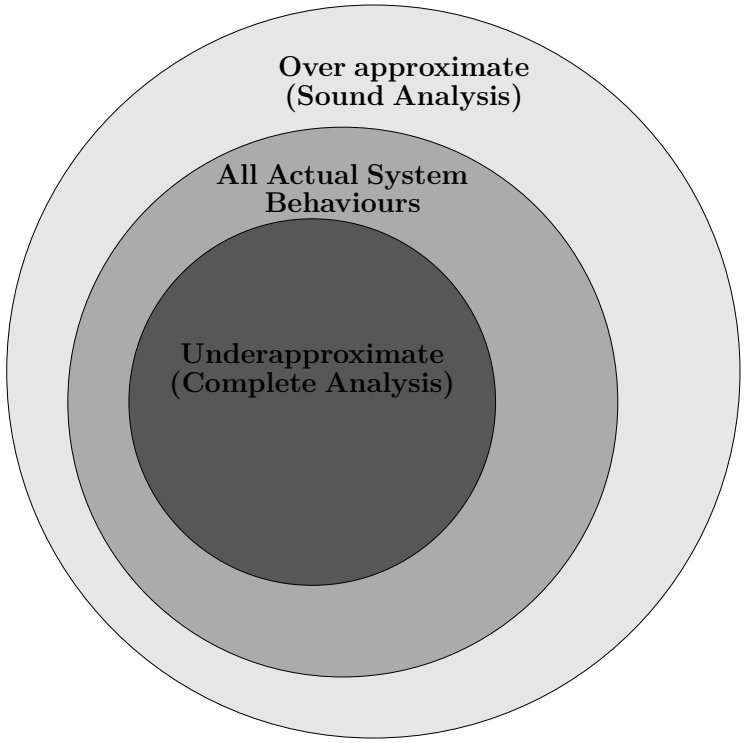


Figure 4.2: *Illustration of the abstraction of system behaviour in static analysis.*

In Figure 4.2 it can be seen how a sound static analysis over-approximates the behaviours of the system. A sound static analyser is guaranteed to identify all violations of a property ϕ , but may also report some false alarms, or violations of ϕ that cannot actually occur. A complete static analysis under-approximates the behaviours of the program. Any violation of a property ϕ reported by a complete static analyser corresponds to an actual violation of ϕ , but there is no guarantee that all actual violations of ϕ will be reported. Note that when a sound static analyser reports no errors, the system is guaranteed not to violate ϕ. This is a powerful guarantee. As a result, most static analysis tools choose to be sound rather than complete.

The development of static analysis dates back to the very earliest compiler development [37] where limited computing power made producing optimal code essential. Here, the original domain in which static analysis was employed, the function was to extract information from programs to facilitate the construction of compilers capable of generating optimal code. The term optimal in this context could entail removing redundant computations or moving loop invariant computations out of loops or any of a wide range of programme optimisations. A wide range of ideas arose from these explorations including *symbolic execution* [157] and *static type checking* [222].

A number of the, initially practical, techniques of static analysis were refined and placed on a more formal basis allowing some of them to be applied to a wider class of models than traditional computer programs [202]. This was achieved through a number of disparate developments. In 1972 Kildall developed a lattice-theoretic foundation of data-flow analysis [156]. This allowed for gathering information about the possible set of values calculated at various points in a computer program by means of efficient *fixpoint* computations. A system's control flow graph is used to determine those parts of a system to which a particular value assigned to a variable might propagate. The analysis is performed by setting up data-flow equations for each node of the control flow graph and solving them by repeatedly calculating the output from the input locally at each node until the whole system stabilizes, i.e. it reaches a fixpoint.

Cousot in 1977 [81] established the relation of static analysis to general system analysis in the form of *abstract interpretation*. This development provided a general method for constructing abstracted operators approximating the semantics of the concrete system in a safe fashion within a chosen abstract domain. Abstract interpretation thus allows reasoning about a system through semantics linked by relations of abstraction, where those abstractions typically are in the form of Galois connections or insertions constructed so that a given property of interest that is present in the concrete system is ensured to be preserved in the abstract domain.

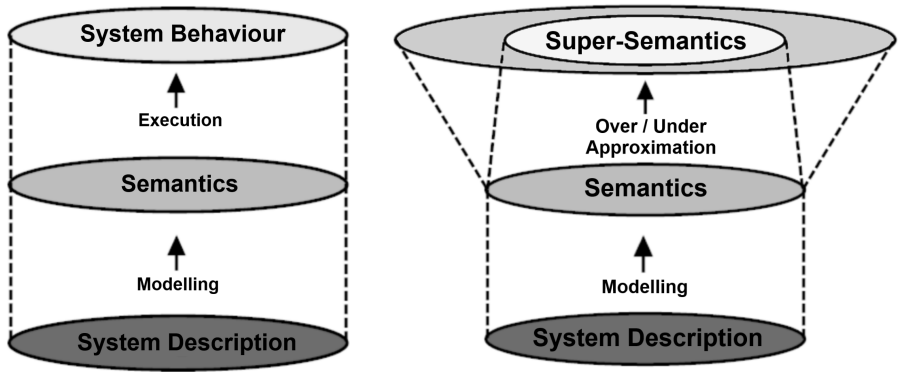
It is generally the technique of abstract interpretation that forms the basis for the application of static analysis techniques outside the traditional complier domain [202] and a wide range of domains have been explored. These domains include more abstract areas such as developing abstract interpretation based static analysis frameworks for process calculi such as Calculus of Communicating Systems (CCS) [205], the $\mu-$calculus [124] or Interactive Markov Chains (IMC) [265]. However, practical applications have also been found outside software verification such as investigation of models of biological systems [203], [223] or validating security protocols [49], [130], [201].

In abstract interpretation, the collecting semantics of a system is expressed as a least fix-point of a set of equations. The equations are solved over some abstract domain that captures the property of interest to be analysed. Typically, the equations are solved iteratively; that is, successive approximations of the solution are computed until a fixpoint is reached. However, for many useful abstract domains, such chains can be either infinite or too long to let the analysis be efficient. To make use of these domains, abstract interpretation theory provides *widening operators*, that attempt to predict the fix-point based on the sequence of approximations computed on earlier iterations of the analysis on a complete lattice. The degradation of precision of the solution obtained by widening can be partly restored by further applying a narrowing operator [82]. However, application of these operations, which is typically necessary when analysing large systems, may produce very large intervals when analysing quantitative properties of interest.

An illustration of ideas of abstract interpretation could be to determine some specific information about a system. Imagine one had a room full of people and one wished to determine "is there a person of age n in the room", keeping a list of all names and dates of births is unnecessary. We may safely and without loss of precision restrict ourselves to keeping a list of the people's ages. If this is too large to handle, we might keep only the age of the youngest m and oldest person M. If n is strictly lower than m or strictly higher than M, then we may safely respond that no such participant was present. Otherwise, we may only be able to say that we do not know.

It is entirely natural that static analysis should be a major consumer of, as well as a testbed and an inspiration for, ideas in semantics because both fields are, at their broadest, concerned with finding formal principles for reasoning about the behaviour of systems. The major distinction is that static analyses are intended to be implemented and must therefore be efficiently computable. By contrast the study of semantics is generally interested in modelling systems as accurately as possible and in proving rather more complex program properties than are considered in static analysis. As illustrated by Figure 4.3(b) the key limitation of static analysis is that the analysis, a super-semantics of a specific semantic system, which encapsulates an underlying system behaviour, is frequently forced to no longer remain true to the underlying system behaviour so as to remain computable. As opposed to employing an analysis technique where, such as what is shown in Figure 4.3(a), the super-semantics perfectly matches the underlying behaviour of a system.

The second key limitation of static analysis is that the key techniques of static analysis do not provide a deductive method for constructing an abstract domain which preserves the properties of interest. While there are a wide range of specific examples of appropriate domains for the analysis for specific properties for specific



(a) System execution (b) Static analysis

Figure 4.3: *Semantic Approximation in Static Analysis*

systems and there exists several ways to combine abstract domains to obtain other more complex abstract domains, this is still a manual process. Further, techniques for constructing analyses that can determine non-functional properties or stochastic systems are immature and frequently lacking tool support.

The best established and most developed tools for static analysis, such as Astrée [48], SLAM [40] and Polyspace [286], are all tied to specific computer programming languages, predominantly C, C++, JAVA and C#. General purpose tools which allow for analysis of abstract models such as those of Stochastic Core BPMN, are immature and lack support for non-functional properties or stochastic behaviour.

Main Strengths:

- Properties of systems can be efficiently computed, even for systems with very large or even infinite numbers of states.
- Static analysis can provide unambiguous answers, e.g. when an over-approximating analysis provides negative result.

Main weaknesses:

- An appropriate abstract domain must be manually defined for each analysis and may not exist.
- Cases exist where an inconclusive result is given, i.e. an analysis may return “maybe” (in practice frequently the case for stochastic properties).

- Tool support is closely tied to specific semantic domain (typically established programming languages). General purpose static analysis where semantics can be arbitrarily defined do not exist.
- Extremely limited examples of analysis of stochastic systems.
- Analysis of quantitative properties may give very large intervals as results.

4.1.3 Model checking

The development of model checking was initially motivated by the problem of employing mathematical logic to reason about programs in a fashion that did not entail theorem proving. Due to the problems encountered when trying to make theorem proving techniques scale up to large programs, model checking sought to develop an approach that avoided the need for proofs.

The crucial innovation which made this work possible was the development by Pnueli [226] of *temporal logics*, a formalism which extends propositional logic with modal/temporal operators describing change over time. Central to this development was the suggestion by Pnueli that this would be well suited to be employed to reason about concurrent systems. Many correctness properties of concurrent systems have a natural description in terms of temporal logic formulae over computation trees, and the easy specification of requirements allowed the effective analysis of complex properties such as invariant properties, liveness properties, and properties of response to an action.

Emerson and Clarke proceed to show that correctness properties can be described using *computation trees* and that from these descriptions fixpoint characterizations can be generated [94]. They proceeded to give conditions on the form of computation tree descriptions to ensure that a correctness property can be characterized using continuous fixpoints. Independently Quielle and Sifakis proposed essentially the same method [236] shortly after.

In the model checking approach to verification, the model constructed is employed to identify the set of all possible future states that the system can be in, and the transitions which can occur between these states. Given a specification in temporal logic [75] which serves to identify the states of interest of the system, automated verification can be performed. In their seminal paper, Clarke, Emerson and Sistla [75] named this approach *model checking* and formulated this as a general method for solving the following problem:

Definition 4.3 (Model Checking Problem)
Given a structure M, a state $s \in M$, and a temporal logic formula f, the model checking problem is to determine:

$$M, s \models f$$

This can alternatively be formulated as: given M and f, calculate the set:

$$\{s : M, s \models f\}$$

Note that the model checking approach is closely bound to the model of interest, even a slightly different model might produce a radically different statespace and determining general theorems of a class of systems is beyond the scope of this method. Further, this approach requires that a statespace can be constructed for the system, which may not be possible in cases when a system has continuous properties or where a clear notion of state does not exist.

Clarke, Emerson and Sistla proceed to address the model checking problem by formulating the temporal logic Linear Temporal Logic (LTL), at the time known as propositional temporal logic, to specify a formula of interest f. They proceed to argue that concurrent programs M can be abstracted to finite state synchronization skeletons, suppressing behaviour irrelevant to concurrency, and provided an LTL model checking algorithm for these that runs in time $O(|f| \cdot |M|^2)$ [75]. LTL was able to describe the properties of individual executions with a semantics defined as a set of executions (traces). For each possible execution of a system which equates to a sequence of events, and this is why it is named "linear time", the satisfiability is checked on the execution with no possibility of switching to another execution path during the checking.

Computation Tree Logic (CTL) developed by Emerson and Clarke [95] provided a different logic for reasoning about systems. CTL describes a *branching-time logic*, meaning that its model of time is a tree-like structure in which the future is not determined; there are different paths in the future, any one of which might be an actual path that is realised. Model checking of CTL properties allowed specifying a formula on all possible execution paths of a system where either all possible runs (the A operator) or only one run (the E operator) are explored when facing a branch.

The development by Hansson and Jonsson of Probabilistic Computation Tree Logic (PCTL) [131] (essentially the same logic was also developed simultaneously by Bianco and Alfaro [46]) provided a probabilistic extension of CTL with the key addition of a probabilistic operator P (Definition A.8, page 253) and quantitative

extensions of CTL's A and E operators in the form of a reward operator R (Definition A.4). PCTL* [46] further extends PCTL by subsuming LTL. This is achieved by removing the need for a path formula to be immediately preceded by the state operator P, this allows any LTL formula to also be a PCTL* formula.

The logic PCTL (and PCTL*) allowed for *quantitative model checking* which extends the standard model checking approach to quantify probabilistic paths through the statespace induced by a Markov Decision Process (MDP) (Definition A.5). In brief, this method operates by transforming an MDP into a Discrete-time Markov Chain (DTMC) (definition A.1) by means of the generation of *adversaries* (Definition A.6), which resolve non-deterministic choice in an MDP thus reducing it to an induced DTMC (Definition A.7). The possible paths through this induced DTMC are explored by means of the temporal logic PCTL (Definition A.2), which is employed to specify specific properties and paths of interest. Hence, in PCTL model checking the P and R operators, which respectively evaluate *probabilities* and *cumulative reward values*, of specific paths as measures over the space of all possible paths of an MDP. The full technical details of this method are given in Appendix A.

Typically in model checking, the structures to be checked are hardware or software systems, and the specification contains safety requirements such as the absence of deadlocks and similar critical states that can cause the system to crash. However, the flexibility of the general approach has allowed model checking to be successfully applied to systems as diverse as:

Biological systems where the specification may express required physical properties of chemical interaction and these are found to agree with observed biological data [133], [182].

Robotics [160] where foraging robot swarms are analysed to ensure properties such as collision avoidance, timing of operations, and power consumption are analysed using probabilistic model checking to verify that global swarm behaviour will indeed function as required.

Sociology [110] where assumptions used in models of the voting patterns for different ethnic groups in New York City, and of the effect of radon on lung cancer in the United States, are checked to determine if they can be falsified by the data upon which they are based.

Tool support for model checking is extensive with several examples of well developed model checking tools which allow fully automated formal verification. The most established of these are SPIN [140], PRISM [170], UPPAAL [43], and

the Failures-Divergences Refinement (FDR) model checker [60]. As an example of the real-world applicability of model checking, the FDR model checker was used in 1996 [178] to discover a flaw in the Needham-Schroeder protocol [197] which had remained undetected despite, extensive manual analysis, for 18 years.

The main caveat of this approach is that as more detail is added to a model the resulting statespace, the set of all possible future states of a system, is prone to combinatorial growth in size. As model checking typically performs an exhaustive search of this statespace verification may become computationally challenging. Known as the statespace explosion, this problem is the main limitation of model checking. However, considerable progress in addressing this increase in complexity has been made and these developments will be discussed in section 4.3.1.

A key strength of model checking is that it functions as a search of the statespace implied by a system for states which satisfy specified properties and, if these properties are violated, this method is able to produce a *counterexample*. This is done in the form of a trace from the initial system state to the state that violates the property being verified. Identifying the exact sequence of actions which must occur for violation of a property to take place also provides useful debugging information when designing business processes. Further, each search of the statespace is unaffected by other searches, meaning that verification of each property of interest can be performed as they are determined to be relevant. Only if the model changes must new verification be performed whereas specification changes only require re-verification of the changed parts of the specification.

Main Strengths:

- Model checking finite statespaces always yields a result.
- Verification inherently determines counter examples in cases when a property is violated.
- Verification properties can be checked one-by-one without any dependencies on other properties to be checked.

Main weaknesses:

- The statespace explosion problem limits the size of models which can be verified.
- Checking only provides results for the specific model in question, no general results are produced.
- Properties to be verified are restricted to temporal logics, which are not as expressive as first order logic.
- Only systems which have a statespace representation can be verified.

4.2 Formal Verification vs. Statistical Simulation

An alternative approach to the verification of concurrent systems is to approximate their behaviour using statistical simulations, such as a Monte Carlo simulation. In general, a statistical simulation works by employing a statistical sampling scheme, where the problem is analysed using a set of randomly generated samples, and measuring what fraction of the random set satisfies a property, in order to determine a property's probability.

When used for analysis of a business process a simulation approach involves developing a model which reflects the behaviour of a process, including the data and resource information, and then performing simulation experiments to better understand the effects of running that process. A number of simulation approaches and corresponding tools have been developed:

- **Protos** [272] is a modelling and analysis tool developed by Pallas Athena and it is mainly applied for the specification of in-house business processes. Processes can be analysed with respect to data, user and control logic perspectives with mean, 90% and 99% confidence intervals of utilization rates, waiting times, service times, throughput times and costs.

- **ARIS** [258] is a professional tool for the dynamic analysis of business processes. Implemented processes are instrumented and recorded in the ARIS Toolset, then used as the input data for business process simulation. The process modelling part supports the definition of business processes represented in Event-driven Process Chains. The simulation results include statistics on events, functions, resources, processes and costs.

- **FileNet** [199] is considered to be one of the leading commercial Business Process Management (BPM) systems. Here a process structure is modelled graphically and tasks are assigned to work queues. Simple arrival patterns of cases are defined, i.e. a fixed number of cases arrive at fixed time points. Historic execution arrival data can also be used. Other performance characteristics can be added manually, but can only have constant values. Both time and cost aspects are taken into account, but without fluctuations because only constant performance measures are used in the simulation. It is possible to create scenarios of a simulation model, but it is not possible to change the process structure in the process simulator itself.

The most prominent distinction between simulation-based verification and formal verification is that the former requires input vectors and the latter does not. The approach in simulation-based verification is first to generate input vectors, and then to derive reference outputs. The thinking process is reversed in the formal verification process. The user starts out by stating what output behaviour is desirable, and then lets the formal checker prove or disprove it. Users do not concern themselves with input stimuli at all.

Although statistical simulation can be useful in some cases, only formal verification is complete, in the sense that it does not miss any point in the input space of a problem. Statistical simulation can explore some situations, but cannot observe all behaviours. Safety properties which guarantee that specific behaviour will always, or, can never, occur need to be evaluated under all possible situations which simply cannot be achieved by the simulation method.

The other key limitation is that simulations need to be executed for a certain amount of time. Whereas verification by model checking is typically unbounded, and can reason about the properties of infinite runs of a system.

4.3 The Choice of Model Checking

In this thesis quantitative stochastic model checking will be employed to analyse and verify business process models of the type described in Section 2.4. This choice is motivated by the following considerations:

1. **Support for quantitative and stochastic properties:** Model checking provides native support for the verification of quantitative and stochastic properties by means of the logic PCTL. In its extended form, PCTL*, support is also available for verification of qualitative properties.
2. **Model size:** While models of business processes can exhibit considerable complexity, their size is relatively modest compared to the complexity of semiconductors [47], [78] or biological systems [133], for which model checking has been successfully employed. Thus model checking is an appropriate choice for models which contain considerably more complexity than those produced by business processes.
3. **Tool support:** A wide range of model checking tools has been developed with some of the most established tools having seen more than two decades of development.

4. **Counterexample generation:** Inherit to the model checking approach is the generation of counter examples when verification properties are violated. When providing design time support for developing business processes this feedback can be extremely useful for debugging and improving processes.
5. **Verification of partial specifications:** Model checking does not require a complete specification in order to perform verification. Properties can be verified independently of each other as they are determined to be relevant for the business process being developed. Once again, this is very useful at design time where a full specification is unlikely to exist. Further, even models for which a specific property can not be verified can still be analysed for other properties.

While the range of properties that can be verified by means of model checking is not as broad as what can be examined using theorem proving, the verification possible is sufficient to analyse precisely those properties desired of business processes. Further the production of counterexamples and support for partial verification are not traditionally possible in theorem proving. Finally, considerable effort is involved to map a description of a business process in any of the common business process languages into a structure that would be amiable to theorem proving . Likewise encoding properties of interest, in particular quantitative and stochastic properties, is considerably more complex in first order logic, most commonly used by theorem provers, compared to expressing such properties in PCTL. Examples of the complexity of model and property specification are well illustrated by the size of models of real-world theorem proving projects such as the formalisation of IEEE Floating Point Arithmetic [86] using *coq* with a total size of 10000 lines of code for 60 definitions and 400 theorems, the verification of a feedback control algorithm for a surgical robot [162] developed in KeYmaeraD [225] involves 156,024 proof steps which are guided by a manually created proof script, and a more extensive model in KeYmaeraD of the European Train Control System cooperation protocol [224] has a total size of over 700MB with 91% of the proof steps able to completed in a fully automatic fashion.

Static analysis is not an appealing approach for analysis of business processes. While this approach involves more complexity in terms of model construction and analysis specification, it is able to scale to considerable larger models than can be tackled with model checking. However, no models encountered in working with the industrial partner or observed in the literature [193], [238] are of sufficient complexity to warrant this approach.

It should be stressed that the distinction between static analysis and model checking is debatable and work by Nielson and Nielson [204] demonstrates how model checking can be cast as static analysis of a modal logics. Conversely work by Schmidt [259] suggests that data flow analysis is model checking of abstract

interpretations. Combined, these efforts show a close relationship between static analysis and model checking, to the extent that one may conjecture that they are reducible to each other. However, for the purposes of this thesis they will be treated as separate approaches with the key distinction being that static analysis is performed on an abstract model of system, whereas model checking seeks to explicitly examine the statespace directly arising from a model.

A key practical consideration in the choice to employ model checking is that it can be automated. Model checking tools generally have a relatively modest learning curve and are able to tackle models of substantial size with limited manual intervention. Compared to well-established tools for theorem proving which often requires considerable configuration to achieve effective results, model-checking tools can be fully automated, allowing their use to be completely hidden from an end-user. Static analysis tools are mostly focused on variation of specific programming languages, in particular C, JAVA and C#. General purpose static analysis tools, similar to theorem proving tools, require considerable manual tuning to be employed effectively.

Being able to perform analysis without a deep understanding of the verification process is essential if it is to be employed by business users. The methods developed in this thesis only require specifications of desired properties for BPMN, and no knowledge of the model checking process itself is required.

4.3.1 The Statespace explosion problem

For clarity in the context of this thesis a statespace will be defined as follows:

Definition 4.4 (Statespace)
The statespace of a discrete system is the set of all possible states of the system. Each coordinate is a vector of state variable, and the values of all the state variables completely describes the state of the system.

In Definition 4.4 each point in the statespace corresponds to a different state of the system. For example a process recording the number of customers in a line would have the statespace $\{0, 1, 2, 3, \cdots\}$

The statespace explosion is the major problem in model checking. The number of global states of a concurrent system with many processes can be enormous. This problem bounds the time-complexity of model-checking algorithms, and

is dependent on the property to be checked and on the size of the transition system. Specifically, the maximal size of the statespace of a transition system with S states and T transitions depends on the presence of:

- **Parallelism** When employing parallelism the statespace of a system is built as the Cartesian product of the local statespaces S_i of the components, i.e. the statespace of the parallel composition of a system with n states and a system with k states yields nk states. The parallel composition of N components of size k produces k^N states.
- **Data variables** In the case when all resources in a model checking problem have a finite domain, e.g. are bounded integers, the number of states in the statespaces grows exponentially in the number of variables in the system. For N variables with a domain of k possible values for the number of states likewise grows up to k^N.

Complexity-theoretic arguments can be used to show that the problem is unavoidable in the worst case (assuming P is different from PSPACE). Fortunately, steady progress has been made over the past 3 decades for special types of systems that occur frequently in practice. In fact, the state explosion problem has been the driving force behind much of the research in model checking and the development of new model checkers. The following key statespace reduction breakthroughs have been made:

Binary Decision Diagram (BDD) representations The main idea behind symbolic model checking is to represent and manipulate a finite state-transition system symbolically as a Boolean function. In particular, Ordered Binary Decision Diagrams (OBDDs) [62] are a canonical form for Boolean formulas that is often substantially more compact than conjunctive or disjunctive normal form, and very efficient algorithms have been developed for manipulating them. Because the symbolic representation captures some of the regularity in the state space determined by circuits and protocols, it is possible to verify systems many orders of magnitude larger than could be handled by the explicit-state algorithms. With the new representation for state-transition systems it has been possible to verify some examples that had more than 10^{20} states [64]. Since then, various refinements of the OBDD-based techniques have pushed the state count up to more than 10^{120} [190].

Partial order reduction One of the most successful techniques for model checking asynchronous systems is the partial order reduction. This technique exploits the independence of concurrently executed events, where two events are independent of each other when executing them in either order results in the same global state. Hence they can be executed in arbitrary order without affecting the outcome of the computation. This

means that, in this case, it is possible to avoid exploring certain paths in the state-transition system. Many specific implementations of these ideas exist such as the persistent sets of Godefroid [116] or the stubborn sets of Valmari [275]. While these differ on the actual details they contain many similar ideas.

Symmetry Reduction The basic idea of exploiting symmetry is that given a statespace and a symmetry group acting on the statespace that preserves transition relations, the symmetry group can be employed to partition the statespace into equivalence classes called orbits. A quotient model is constructed that contains one or more representative from each orbit. This partitioned statespace will in general, be much smaller than the original statespace. First introduced by Clarke et. al. in 1996 [77], this technique has seen extensive development and can be combined with BDDs to achieve substantial reductions in their size.

CEGAR Counter example guided abstraction refinement attempts to prove the properties on a system by first simplifying it. The approach begins checking with a coarse (imprecise) abstraction of an model and iteratively refines it [76]. When a violation (counterexample) is found, it is analysed for feasibility (i.e., to determine if the violation genuine or the result of an incomplete abstraction). If the violation is feasible, it is reported to the user; if it is not, the proof of infeasibility is used to refine the abstraction, and checking begins again. In general, the presence of spurious counterexamples cannot be avoided, since the abstract model over-approximates the state space of the concrete system. This is due to the loss of information caused by the abstraction mapping. However, the state space of the abstract system is usually much smaller than that of the concrete system, making the abstract system amenable to model checking.

Bounded model checking Given a finite state-transition system, a temporal logic property, and a bound k, bounded model checking generates a propositional formula that is satisfiable if and only if the property can be disproved by a counterexample of length k [47]. This propositional formula is then fed to a Boolean satisfiability (SAT) solver. If no counterexample of length k is found, then the property is considered to hold. For safety properties (i.e., checking whether a "bad" state is unreachable), it can be shown that we only need to check counterexamples whose length is smaller than the diameter of the system i.e. the smallest number of transitions to reach all reachable states. Note, that this approach to model checking is an approximate method and does not provide exact results.

Business processes, as formally defined in Chapter 3, have in this thesis been constructed so as to limit their complexity and hence the size of their implied statespace. Typical models of business processes consist only of hundreds to hundreds of thousands of states producing state spaces which are quite

manageable. They make limited use of parallelism mostly in the form of simple message passing between separate processes. However, the approach to modelling business processes does grows models statespace significantly in the case of bounded rewards, when terminal states for reward exhaustion must be modelled and are inherently bounded in scope. In this case the models will however exhibit a large degree of symmetry and symmetry reduction techniques are able to reduce the effective complexity of this to little more than an equivalent non-bounded model. Note, that properties captured as rewards (Appendix A.4), such as time or money spent, are evaluated as the statespace is traversed and do not add to the size of the underlying statespace.

Scaling of the specific analysis methods presented in this thesis is explored in Section 9.3.

4.4 Probabilistic Model Checking Tools

This section focuses on model checking tools which are suited to the analysis of quantitative probabilistic systems. The basic operation of these tools is shown in Figure 4.4. In general, the inputs (highlighted in light grey) are a probabilistic model and a property specification defined in appropriate modelling languages. The outputs (highlighted in dark grey) is either confirmation or the value of a queried reward. In the case when a property is violated, most tools allow for a counterexample to be provided.

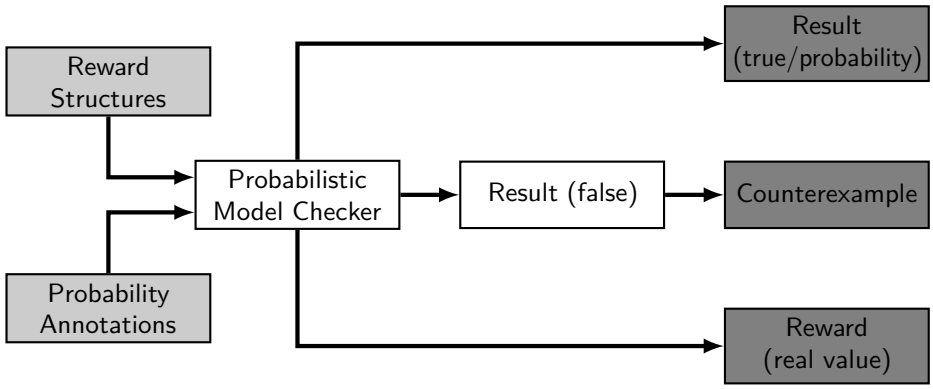


Figure 4.4: *The basic design of quantitative probabilistic model checking tools.*

There are currently four main well developed probabilistic model checking tools in active development:

1. PRISM [170] has been developed at the University of Oxford (UK). It is a free and open source tool, distributed under the GNU General Public License (GPL). The tool is developed using a combination of Java and C++. Its user interface and parsers are written in Java.
2. Markov Reward Model Checker (MRMC) [151] *Markov Reward Model Checker* has been developed at Aachen University (Germany). It is distributed under the GNU General Public License (GPL). MRMC is a command-line-based tool, implemented in C and currently only supports the Linux platform.
3. MOdest TOol EnviRonment (MOTOR) [50] has been developed at Saarland University (Germany). It is distributed free of charge for non-commercial use, and developed under a closed-source license. It is developed in C++ and provides a graphical user interface written in C#.
4. UPPAAL-SMC [63] The name of this tool is derived from the first three letters of Uppsala University (UPP) and Aalborg University (AAL); while SMC stands for the Statistical Model Checking extension of this tool. It is developed in collaboration between these two universities and is distributed free of charge for non-commercial use and developed under a closed-source license. It is developed in C++ and provides a graphical user interface written in Java.

In Chapter 3, the modelling of business processes in BPMN is achieved by means of process graphs which are extended with both non-deterministic and probabilistic behaviour, and annotated with rewards. This motivates the need to employ a model checking tool which is able to operate on Markov decision processes, or an extension of these. An overview of the capabilities of each of these tools in terms of support for probabilistic models formalisms, property specification logics, and rewards is shown in Table 4.1.

Tool	**Models**	**Properties**	**Rewards**
PRISM	DTMC, CTMC, MDP, PTA	PCTL, CSL, LTL, PCTL*	Yes
MRMC	DTMC, CTMC	PCTL, CSL	Yes
MOTOR	MODEST (MDP, DTMC, CTMC, PTA, LTS, SHA)	(Special)	Yes
UPPAAL-SMC	PTA	WMTL	No

Table 4.1: *Overview of the capabilities of the main quantitative probabilistic model checking tools.*

From Table 4.1 it is clear that the main candidates for analysis of business processes are PRISM and MOTOR. UPPAAL-SMC, while supporting Probabilistic Timed Automata (PTA), an extension of Markov decision processes with clocks and constraints on clocks, does not support rewards when analysing these models, significantly reducing what can be learned about business processes from this tool. MRMC does not support Markov decision processes and as such is not suited.

MOTOR is a powerful and appealing tool, built on top of PRISM and based upon the MoDeST (MOdeling and DEscription language for Stochastic Timed systems) language. This language, in addition to being able to describe MDPs, DTMCs and PTAs, is also able to describe *stochastic hybrid automata* (SHA), which combine nondeterministic choices, continuous system dynamics, stochastic decisions and timing, and real-time behaviour, including nondeterministic delays. While powerful, this formalism is considerably more complex than a MDP, and the additional features it provides are not relevant for the description of business processes driven by the considerations in Section 1.1.

Due to the unsuitability of the previously mentioned tools and due to the generally high-performance of PRISM, the PRISM model checker is a strong candidate for a tool to be employed for the formal analysis of business processes. The support for a wide range of temporal logics for property specification combined with support for a wide range of Markov models including MDPs mean PRISM is able to perform extensive analysis of business processes which exhibit both nondeterminism and stochastic behaviour. Further PRISM supports a wide range of methods to mitigate the statespace explosion problem allowing it, in general, to tackle models of greater size than the competition. PRISM Is described in more detail in the following section.

4.4.1 PRISM

First developed as part of his PhD Thesis by David Parker [218], the PRISM tool has seen extensive development and has grown to be a very capable model checker. A formal presentation of the general mathematical theory of how PRISM operates is outlined in Appendix A.

PRISM is an open-source software tool which accepts probabilistic models written in a textual PRISM modelling language based on *Reactive Modules* [30], a state-based language using guarded commands. The three main model types supported are DTMCs, MDPs and PTAs, and properties for their analysis can be specified using the PRISM specification language which implements the probabilistic

temporal logics, PCTL, Continuous Stochastic Logic (CSL) (for Continuous Time Markov Chains (CTMCs), LTL and PCTL*. The architecture of PRISM is shown in Figure 4.5.

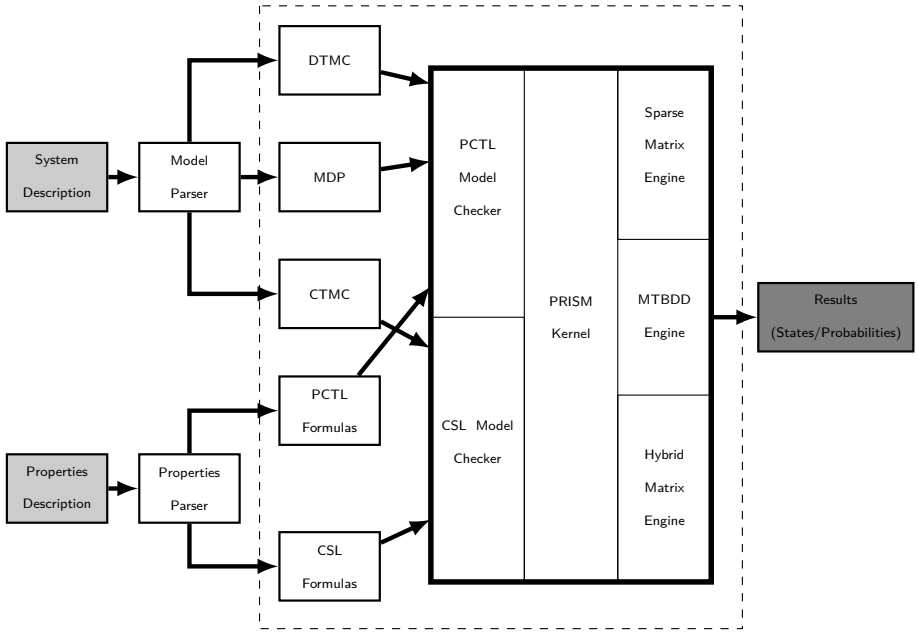


Figure 4.5: *The architecture of the PRISM model checker. The tool itself is shown within the dashed area (source [166]).*

PRISM includes multiple model checking engines, several of which are based on symbolic implementations (using binary decision diagrams and their extensions). These can enable the probabilistic verification of models of up to 10^{10} states [169], however the tool achieves optimal performance on models in the range of 10^4 to 10^8 states. To achieve this, PRISM offers the user a choice between the following three data structures, with regard to MDPs, for model checking [166]:

1. **Sparse Matrix** This is an explicit representation of the statespace, and can be a good option for smaller models where model checking takes a long time. For larger models, however, memory usage quickly becomes prohibitive. As a rule of thumb, the upper limit for this engine, in terms of model sizes which can be handled, is about a factor of 100 less than the hybrid engine.

2. **Multi-Terminal Binary Decision Diagrams** (MTBDDs) are extensions of BDDs which can encode quantitative data. This compact representation is much more unpredictable in terms of performance but, when a model exhibits a lot of structure and regularity, can be very effective. This engine has been successfully applied to extremely large models (up to 10^{10} states), and can be used in cases where the other two engines cannot be applied.

3. **Hybrid** The hybrid engine is enabled by default in PRISM. It uses a combination of symbolic and explicit data structures as used in the Multiple Terminal Binary Decision Diagram (MTBDD) and sparse engines, respectively. In general, it provides the best compromise between checking time and memory usage. It almost always uses less memory than the sparse engine, but is typically slightly slower. The size of model which can be handled with this engine is quite predictable. The limiting factor in terms of memory usage comes from the storage of 2-4 arrays (depending on the computation being performed) of 8-byte values, one for each state in the model.

All engines perform the same calculations; therefore the choice between MTBDD, sparse matrix, or hybrid representations does not affect the results of the model checking. The accuracy and the speed of convergence on a result depends on the chosen iterative method to solve the systems of linear equations as described in Appendices A.3 and A.7. PRISM offers a choice between several well established methods:

- Gauss-Seidel (also backwards).
- Jacobi method.
- Jacobi Over Relaxation (JOR) (also backwards).
- Power iteration.
- Successive over-relaxation (SOR).

Common to all of these methods is the way that PRISM checks convergence, i.e. decides when to terminate the iterative methods because the answers have converged sufficiently. This is done by checking when the maximum difference between elements in the solution vectors from successive iterations drops below a given threshold. The default value for this threshold is 10^{-6}, but it can be altered. Also, the maximum number of iterations performed can be given an upper limit in order to trap the cases when computation will not converge, here the default limit is $10,000$.

A variety of advanced capabilities are also available:

Simulation engine [171] A discrete-event simulation engine is included in PRISM for debugging models and to support so-called statistical model checking techniques.

Optimal adversary generation [171] PRISM's MDP verification has the ability to generate optimal adversaries. This means that, when PRISM computes the minimum or maximum value for a probabilistic reachability (or expected reward) property, it can also generate a resolution of non-determinism in the model that produces it.

Furthermore, PRISM implements the following techniques, in addition to the possible statespace representations, to further combat the statespace explosion problem:

Symmetry Reduction PRISM employs component symmetry, in which any pair from a set of symmetric components in a model can be exchanged with no effect on the overall behaviour. Verification of a model can then be employed on on a bi-similar quotient model which is, at the upper bound, factorially smaller [90].

Partial order reduction PRISM combines the stubborn set approach and probabilistic model checking [129]. PRISMadopts the weak stubborn set method, instead of the well-known (strong) stubborn set, to obtain more compact reduced models.

CEGAR A quantitative variant of CEGAR is implemented in PRISM in the form of an extensible tool-kit, with support for multiple model types, refinement strategies, and configurable optimisations [152]. Although these require some manual tuning, they can greatly expand the complexity of models that can be feasibly checked.

4.5 Using the PRISM Model Checker

The underlying theory of how the PRISM model checker performs model checking of MDP structures is outlined in Appendix A. The basic design of the PRISM model checker is described in Section 4.4.1. This section describes the PRISM modelling and property specification languages. These are used as inputs to the PRISM model checker and are the target of translation for stochastic Core BPMN models.

4.5.1 The PRISM Modelling Language

PRISM models are expressed in a guarded command language which is based upon the formalism of reactive modules [30]. While PRISM supports a number of different types of models, only the modelling of MDPs is considered here, due the nature of the combination of probabilistic and non-deterministic branching allowed by definition of exclusive gateways in Definition 3.14.

MDPs are defined in the PRISM modelling language by composing a number separate of modules which interact with each other by means of message passing. An individual module takes the following form [217]:

```
module
x : [0..n] init 0;
[act] guard → p_1 : update_1 +...+ p_n : update_n;
endmodule
```

The `module ... endmodule` construct defines a *module* which contains two parts: *variables* and *commands*. The variables describe the possible states that the module can be in, e.g. (`x :[0..2] init 0`) defines a variable `x` which ranges from 0 to 2 and starts with a value of 0.

Commands describe the behaviour of the modules, i.e. the way in which their states change over time. Inside the command, the *guard* (`guard`) is a predicate over all the variables in the model (including those belonging to other modules), e.g. (`(x=1)&(t!=3)`). Following the *guard*, the command defines one or more *transitions* (`p_i :update_i`), where each update describes a transition the module may perform if the guard is true. A *transition* consists of one or more *updates* (`update_i`) which are individually specified by giving the new values to the variables of the module, and an associated probability (`p_i`), e.g. (`0.2:(x'=2)`). Elements of the updates may be concatenated with & and each element must be bracketed individually. If an update does not give a new value for a local variable, the variable is assumed not to change.

A command may be prefixed by an *action* label inside square brackets, e.g. (`[act]`) which, in the style of many process algebras, is a *synchronisation label*, allowing commands to be synchronised and which forces two or more modules to make transitions simultaneously if the transition is enabled. By default, all modules are composed in the style of standard Communicating Sequential Processes (CSP) [138] parallel composition, i.e. modules synchronise over all their common actions. However, it is possible to define precisely the way in which the set of modules are composed in parallel. This is specified using the

system ... **endsystem** construct, placed at the end of the model description, which should contain a process-algebraic expression. This expression should feature each module exactly once, and can use the following CSP based operators:

- `M_1 || M_2` *parallel composition* of modules `M1` and `M2` synchronising on only actions appearing in both `M1` and `M2` (the default behaviour).
- `M1 |[a,b,...]| M2` *restricted parallel composition* of modules `M1` and `M2`, synchronising only on actions from the set {`a, b,...`}.
- `M /{a, b,...}` *hiding* of actions {`a, b,...`} in module `M`.
- `M{a<-b,c<-d,...}` *renaming* of actions `a` to `b`, `c` to `d`, etc. in module `M`.

When evaluating the expression, the hiding and renaming operators bind more tightly than the three parallel composition operators. No other rules of precedence are defined and parentheses should be used to specify the order in which modules are composed.

Rewards are declared in a similar fashion to modules by the following construct:

```
rewards name
[act] guard : reward
endrewards
```

Where `name` is a label for the reward structure, the body of the reward structure is interpreted as the transitions from states which satisfy the guard `guard` (a predicate over all the variables of the model) and are labelled with the action `act` increment of the reward structure `name` with the positive real value of the reward `reward`. If no action is specified, the reward is allocated in all states which satisfy the guard. For example, the reward `x=0 :100` would denote that in the state when `x` is 0 the given reward is incremented by 100 and would correspond to a *node reward*. States with an explicitly defined action label are employed to identify *transition rewards*, where for example, the reward `[treat]:1` within a reward structure modelling drugs used in a medical process would denote that transitions labelled `[treat]` will increment the reward structure by 1.

4.5.2 Semantics Imposed by PRISM

To perform analysis of stochastic core BPMN models, they must be mapped to the PRISM modelling language which is, in essence, a representation of a MDP [282]. However, the structure of the PRISM modelling language differs significantly from the traditional graph-based representation of MDPs. Hence, in the generation of the state space implied by the model, PRISM imposes a semantic interpretation on the execution of a business process captured as

a process graph. This effectively treats the input model as a discrete set of states representing possible configurations of the system. Transitions in the business process are modelled as probabilistic state transitions combined with non-deterministic choices between several discrete probability distributions over successor states from a given initial state.

PRISM is a tool which seeks to construct the entire statespace (i.e. all possible reachable states) implied by a set of interacting modules. This is achieved in broadly the same fashion defined for reactive modules [30] upon which the PRISM modelling language is based. Here the semantics are not defined in a compositional manner, i.e. by first giving the semantics of each module in the system and then combining these results. The reason for this is that guards (and updates) of one module are allowed to refer to the variables of other modules (and indeed global variables). Instead, the semantics of a PRISM model are defined by translating it's set of modules into a single *system module* in a compositional manner and then defining the semantics for the whole system through this single module.

The process of constructing the master system module from its component modules is by composition of the modules which are each defined by a reactive modules [30] style process-algebraic expression, which can include *parallel composition* of modules, *action hiding* and *action renaming*. Note that, in this construction process, it is required that all updates of all commands have been expanded to explicitly include all local variables of the module and all global variables, even those that do not change.

Given modules M_1 and M_2 which interact via a set of partially common actions A, the process of constructing the system model begins by addressing parallel composition. Although there are three types of parallel composition, one need only consider the case of *restricted parallel composition* of modules $M_1|[A]|M_2$ when M_1 and M_2 interact via some elements of A. This is due to the fact that since $M_1 \parallel M_2$ is equivalent to $M_1|[\emptyset]|M_2$ and $M_1 \parallel M_2$ is equivalent to $M_1|[A_1 \cup A_2]|M_2$ where A_i is is the set of actions that appear in module M_i. The commands of the system module $M = M_1|[A]|M_2$ are constructed according to the following rules:

1. For each of the commands:

$$[]\ g \rightarrow p_1 : u_1 + \ldots + p_n : u_n \text{ of } M_1$$
$$[]\ g \rightarrow p_1 : u_1 + \ldots + p_n : u_n \text{ of } M_2$$

   Add:

$$[]\ g \rightarrow p_1 : u_1 + \ldots + p_n : u_n \text{ to } M$$

2. For each $a \notin A$ and each of the commands:

$$[a]\ g \to p_1 : u_1 + \ldots + p_n : u_n \text{ of } M_1$$
$$[a]\ g \to p_1 : u_1 + \ldots + p_n : u_n \text{ of } M_2$$

   Add:

$$[a]\ g \to p_1 : u_1 + \ldots + p_n : u_n \text{ to } M$$

3. For each $a \in A$ and commands:

$$[a]\ g \to p_1 : u_1 + \ldots + p_n : u_n \text{ of } M_1$$
$$[a]\ g' \to p'_1 : u'_1 + \ldots + p'_n : u'_n \text{ of } M_2$$

   Add:

$$\begin{aligned} [a]\ g\ \&\ g' \to \quad & p_1 * p'_1 : u_1 \& u'_1 + \ldots + p_n * p'_1 : u_n \& u'_1 \\ + \quad & p_1 * p'_2 : u_1 \& u'_2 + \ldots + p_n * p'_2 : u_n \& u'_2 \\ & \vdots \\ + \quad & p_1 * p'_n : u_1 \& u'_n + \ldots + p_n * p'_n : u_n \& u'_n \text{ to } M \end{aligned}$$

With regard to action hiding, the commands of $M = M'/A$ are constructed according to the following steps:

1. For each command:

$$[]\ g \to p_1 : u_1 + \ldots + p_n : u_n \text{ of } M'$$

   Add:

$$[]\ g \to p_1 : u_1 + \ldots + p_n : u_n \text{ to } M$$

2. For each $a \notin A$ and command:

$$[a]\ g \to p_1 : u_1 + \ldots + p_n : u_n \text{ of } M'$$

   Add:

$$[a]\ g \to p_1 : u_1 + \ldots + p_n : u_n \text{ to } M$$

3. For each $a \in A$ and command:

$$[a]\ g \to p_1 : u_1 + \ldots + p_n : u_n \text{ of } M'$$

   Add:

$$[]\ g \to p_1 : u_1 + \ldots + p_n : u_n \text{ to } M$$

Action renaming for the commands of $M = M'\{a_1 \leftarrow b_1, ..., a_m \leftarrow b_m\}$ is handled as follows:

1. For each command:

$$[]\ g \rightarrow p_1 : u_1 + \ldots + p_n : u_n \text{ of } M'$$

   Add:

$$[]\ g \rightarrow p_1 : u_1 + \ldots + p_n : u_n \text{ to } M$$

2. For each $a \notin \{a_1, ..., a_m\}$ and command:

$$[a]\ g \rightarrow p_1 : u_1 + \ldots + p_n : u_n \text{ of } M'$$

   Add:

$$[a]\ g \rightarrow p_1 : u_1 + \ldots + p_n : u_n \text{ to } M$$

3. For each $1 \leq i \leq m$ and command:

$$[a_i]\ g \rightarrow p_1 : u_1 + \ldots + p_n : u_n \text{ of } M'$$

   Add:

$$[b_i]\ g \rightarrow p_1 : u_1 + \ldots + p_n : u_n \text{ to } M$$

Having defined the contents of system module by means of the previous construction rules, the system module semantics can be defined, given:

- C a multiset of commands generated by the rules above.
- $V = v_1, ..., v_m$ is the set of variables, both local and global, that appear in the system description.

The state space of the system, which is equivalent to an MDP, is constructed in the following fashion. A state is a tuple $(x_1, ..., x_m)$ where x_i is a value for the variable v_i. The set of all states S is therefore the set of all possible valuations of the variables in V that are permissible by the system.

The set of initial states can be specified in one of two ways: either by giving an initial value for each variable, or by assigning a predicate over variables (using the **[init ... endinit]** construct). In the former case, $\bar{S} = \bar{s}$ where $\bar{s} = (\bar{x}_1, ..., \bar{x}_m)$ and $\bar{x}_i$ is a the initial value of the variable v_i (if the initial value of a variable is left unspecified, it is taken to be the minimum value of the variable's range). In the latter case, $\bar{S}$ is the subset of states S which satisfy the predicate specified in the **[init ... endinit]** construct.

The semantics for a single command of the system module can now be defined. From this point, any action-labels assigned to commands in C can be ignored; these were required only for the process-algebraic system module construction and can be safely discarded. Hence, each command $c \in C$ takes the form:

$$[]\ g \rightarrow p_1 : u_1 + \ldots + p_n : u_n$$

Since the guard g is a predicate over the variables in V and each state of the system is a valuation of these variables, a command c's guard g defines a subset of the global state space S i.e. $S_c = \{s \in S | s \models g\}$. Each update u_j of c corresponds to a transition that the system can make when in a state $s \in S_c$. The transition is defined by giving the new value of each variable as an expression. Hence, one can think of u_j as a function $u_j : S_c \rightarrow S$. If u_j is $(v'_1 = expr_1) \wedge \ldots \wedge (v'_m = expr_m)$, then for each state $s \in S_c$:

$$u_j(s) = (expr_1(s) \ldots expr_m(s))$$

Using the probability value p_j associated with each update u_j, the command c defines, for each $s \in S_c$, a function $\mu_{c,s} : S \rightarrow \mathbb{R} \geq 0$ where for each $t \in S$:

$$\mu_{c,s}(t) \stackrel{def}{=} \sum_{1 \leq j \leq n \wedge u_j(s) = t} p_j \tag{4.1}$$

Note that, for MDPs, the syntactic constraints placed on the constants p_j mean that the function $\mu_{c,s}$ is actually a probability distribution over S. One can now define the *transition probability function* itself, namely the function *Steps* of an MDP (see Definition A.5). This function $Steps : S \rightarrow 2^{Dist(S)}$ is such that for any $s \in S$:

$$Steps(s) = \{\mu_{c,s} | c \in C \wedge s \in S_c\} \tag{4.2}$$

The construction of *system module* and *Steps* the *transition probability* function allows for determination of the entire statespace of a set of PRISM modules in the form of an MDP by iterative application of μ to initial state of the model.

The translation of BPDs to PRISM code, described in Section 5.4 allows for each state in the resulting PRISM statespace to correspond to a configuration of Core BPMN model. Each statespace state will have associated values of reward structures and a probability of being visited that correspond to a given Core BPMN model configuration, after execution of a specific trace, as defined in Section 3.5. Hence, a Core BPMN model which has, for example, executed trace $< A, B, A >$ would be denoted in the PRISM statespace as a unique state which is reachable through states in the PRISM state space that correspond to executing

first A then B and then returning to state A. This point in the statespace would have associated variables V which denote the reward values associated with this specific path. In addition the probabilities of arriving in a sate s though possible paths to s can be determined by evaluating $Steps(s)$. In total the PRISM statespace records all possible configurations a Core BPMN model can attain when executed. Note that the statespace optimisation techniques described in Section 4.3.1, which are implemented in PRISM may compact this statespace in various ways but will preserve the representation of evolution of qualitative and quantitative properties of the model with respect to a specific query of a model.

A key consequence of this approach to statespace generation is that in the semantics imposed by PRISM, concurrent elements of a business process do not execute synchronously, instead these transitions are instead arbitrarily interleaved in time (in the style of CSP [138] parallel composition). PRISM will generate all possible interleavings, as dictated by message passing, of separate BPDs and all possible interleavings within each BPD of all Core BPMN elements between parallel fork and merge gateways. An illustration of this interleaving behaviour for the concurrent part of the Core BPMN example process given in Section 3.4 is shown in Figure 4.6.

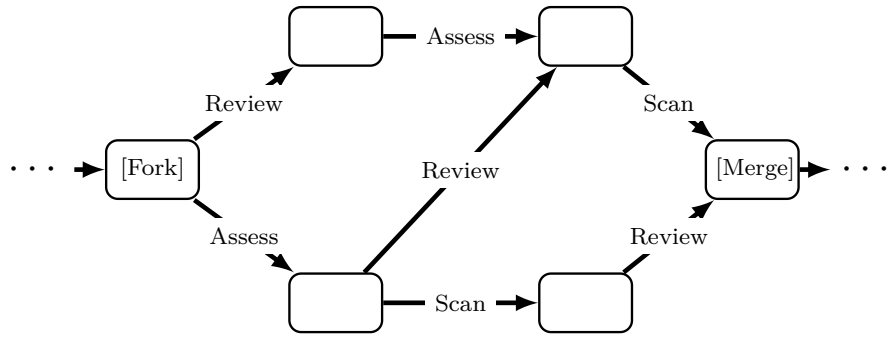


Figure 4.6: *Illustration of interleaving semantics imposed by the PRISM model checker (extract of the state space of the parallel processes of the example from Section 3.4).*

Note how the possible paths from the [fork] gateway to the [merge] gateway cover all possible allowed interleavings of assessing and then scanning a patient, and examining the patient's medical record. This maximal interleaving ensures that all possible execution paths are explored, but does not capture so called "true concurrency". This semantic distinction is expressed by Baeten [38] by means of the *the total order assumption* which requires, for the assumption to hold that "all execution sequences of observable actions or events are totally ordered by precedence". This assumption is shown by Baeten to be well suited

to cases when an execution sequence is totally ordered by time and a totally ordered set, such as the non-negative real numbers is employed to model the time domain. Furthermore, when business processes are managed by BPM software, this multitasking, or multiprocessing, is enabled by a software or hardware scheduler. In this type of scheduling the actual execution semantics of a business process will have the total order assumption imposed upon their execution. This assumption ensures all execution sequences that will be possible in practice are present in a model generated by PRISM. Therefore verification of a system under the total order assumption closely matches its actual real world behaviour and verification of systems under the total order assumption provides a strong guarantee of their safety properties.

However, it is this interleaving behaviour which leads to the requirement that, when evaluating performance properties in the form of probability and reward queries, only upper or lower bounds on the expected values can be determined. Note that in a system where there is no non-determinism and only probabilistic behaviour, no interleaving behaviour is present and the bounds will converge, as this case, in essence, reduces to DTMC model checking [39].

In addition, PRISM imposes a semantics which does not explicitly allow *deadlock* states and, on execution, any model that features states which deadlock will have an additional self-loop added to the state. The analysis of models for deadlock states is possible, however, as the process of adding self-loops to deadlocking states also marks them as originally producing deadlocks. The semantic interpretation imposed on Core BPMN by PRISM with regard to deadlocks allows other parts of a BPMN model to continue to execute and is, in the case of BPMN, in line with the extensive work of Wong and Gibbons [289] when encountering a deadlock or end event. The same behaviour is modelled here, and this would seem to align with real-world cases where a single deadlocking process would not lead to other business processes halting their execution, as they are not aware of other processes' deadlock condition.

Based on the mechanism of *process graph* synchronisation described Section 3.2.1, and developed as the denotational execution semantics of Core BPMN in Section 3.5 the semantics imposed during analysis of a Core BPMN model are the totality of possible *traces* of the model. Hence the analysis treats synchronization as the direct incorporation of processes which communicate with each other as a single larger model, for which a full statespace is determined.

For the purpose of the analysis of Core BPMN models, it will be shown in Section 5.4 how these can be translated into PRISM models. Hence the execution semantics of Core BPMN models, as stated in Section 3.1, are defined through this translation. While imposing a semantics on Core BPMN in this fashion may seem somewhat convoluted it is motivated by two considerations. Firstly, as

established in Section 2.3.2.2, the semantics of BPMN are poorly defined and this approach allows great freedom in assigning a semantic understanding of models and then consequently exploring the totality of the resulting execution through model checking. Secondly, Objective 2 is to allow analysis of business process models, which is enabled by this approach as it allows the effective checking of *safety properties* through exploring all possible executions.

It should be noted, that the model checking approach examines all possible executions by expanding all possible resolutions of non-determinism, as such during the analysis stage non-determinism is resolved using all possible adversaries as described in Appendix A.6.

4.5.3 PRISM Property Queries

PRISM's property specification language, as it relates to MDPs, subsumes the logics LTL [39] and PCTL [131] (detailed in Appendix A.2) in the form of PCTL* [46]. In fact, PRISM also supports numerous additional customisations and extensions of this logic. When performing analysis of an MDP using PRISM, a statespace for the model in question is generated, and PRISM's property specifications are then used to define a query about states or paths of interest within the statespace.

The P operator, as it relates to PCTL checking of MDPs, is formally defined in Definition A.8. Its implementation in PRISM comes in two variants $\mathsf{P}_{\mathsf{min}} = \mathsf{n}$ and $\mathsf{P}_{\mathsf{max}} = \mathsf{n}$ which denote respectively determining the lower or upper bounds of a probabilistic query, determined by the resolution of non-determinism of the MDP. If $n \in [0, 1]$ is explicitly defined, the result of model checking is to determine if the property ψ is true or false, i.e. $\mathsf{P} : \psi \rightarrow \{true, false\}$. However, n may be defined as ? in which case PRISM determines a probability, i.e. $\mathsf{P} : \psi \rightarrow [0, 1]$.

One of the most fundamental tasks when specifying properties of a model is to identify particular states or sets of states with the PRISM statespace to which a path, and associated probability bounds and reward values, can be determined. For example, to verify a property such as *the business process eventually terminate successfully with probability* 1, it is first necessary to identify the states of the model which correspond to situations where *the business process has terminated successfully*. In terms of the way temporal logics are usually presented, these correspond to atomic propositions defined for MDPs in Definition A.5. In PRISM, this is achieved simply by declaring an expression in the PRISM query language which evaluates to a Boolean value, here traditional Boolean operators may be used to combine expressions. This expression will typically contain references to

variables (and constants) from the model to which it relates. The set of states corresponding to this expression is those for which it evaluates to true. A state expression is said to be *satisfied* in those states where it equates to true.

A practical extension of the PCTL language supported by PRISM is the use of *labels*. These are a way of identifying sets of states that are of particular interest. Labels can only be used when specifying properties but, for convenience, can be defined in model files as well as property files. For example the expression **label** `"failure" = temp>100&alarm=false` would identify a set of states where the Boolean expression `temp>100&alarm=false` equates to true, and these are identified by means of the label `failure`. Likewise **formula** expressions can be used as shorthand for the evaluation mathematical expressions, e.g. **formula** `num_tokens = q1+q2+q3+q+q5` will evaluate the sum $q1+q2+q3+q+q5$ at any point it used in a query expression.

When employing the P operator, the PRISM property specification language allows for the definition of paths by means of the PCTL path properties (see Appendix A.2) to define a path to an identified set of states. A path formula to these states is *satisfied* when the statespace contains such a path. In terms of MDPs, PRISM allows for the use of the following path operators:

- Xa The *next* operator is a unary operator that specifies for a path that a given property a holds in the path's next state.
- aUb The binary *until* operator specifies that, for a given path, in some state of the path, the property b is true and in all preceding states the property a is true.
- Fa The unary *eventually* operator specifies that, for a given path, a eventually becomes true at some point along the path.
- Ga The unary *always* operator specifies that, for a given path, a is true in all states along the path.

PRISM also supports probabilistic model checking of the temporal logic LTL (as it is subsumed within PCTL*). LTL provides a richer set of path properties for use with the P operator by permitting temporal operators to be combined. Further, PRISM state and path identification expressions can combined in arithmetic expressions and by means of traditional propositional logic operators (negation, conjunction and disjunction), to allow more complex measures than standard PCTL* to be expressed.

The other main PRISM MDP property query operator is the R operator which is defined in the same way as the P operator in Definition A.8, with respect to resolving the non-determinism of MDP as a DTMC [282], and determining the accumulated reward value in the same fashion as the probability of a specified

path. Again, the implementation in PRISM comes in two variants $\mathsf{R}^{\mathtt{name}}_{\mathsf{min}}$ and $\mathsf{R}^{\mathtt{name}}_{\mathsf{max}}$ which denote the determination of the lower or upper bounds, determined by the resolution of non-determinism of the MDP for a specific reward structure denoted by `name`. For a given path ψ the reward operator is a mapping $\mathsf{R}_{\mathsf{n}} : \psi \rightarrow \{true, false\}$ when $n \in [0, \infty[$ and if n is defined as ? the mapping is to $\mathsf{R}_{\mathsf{n}} : \psi \rightarrow [0, \infty]$. Due to the probabilistic nature of an MDP, the R operator determines the bound on the expected value of a random variable associated with a particular reward structure.

When performing R operator based queries of an MDP derived from a business process, PRISM allows determining *reachability reward* properties. These associate a reward with each path of a model, and refer to the rewards accumulated along a path until a certain point is reached. This point is defined in the same fashion as for the P operator by means of identifying states and the paths to these states.

Examples of PRISM properties queries, that, for example, could relate to a medical robot, are:

- For example, a boolean variable *Contaminated*, which could either be a simple variable or a more complex expression over other state variables, could describe whether a dose of drug is contaminated. A query of the lower bound on the probability that the dose is contaminated is expressed as:
$$\mathsf{P}_{\mathsf{min}} = ?[\mathsf{F}\ \mathit{Contaminated}]$$
- Consider a system comprising two components, A and B, each of which can be contaminated independently. That the upper bound on the probability that component B is contaminated before component A is less than 0.1 can be expressed as:
$$\mathsf{P}_{\mathsf{max}} = 0.1[(!\mathit{Contaminated}_A)\ \mathsf{U}\ \mathit{Contaminated}_B]$$
- Determining the upper bound on the probability that the system globally does not enter states defined as *contaminated* can be expressed as:
$$\mathsf{P}_{\mathsf{max}} = ?[(\mathsf{G}\ \neg \mathit{Contaminated}_A]$$
- This example represents determining the upper bound of the mean-time to having a dose of drug ready, identified as the states *DoseReady*, of a system with a defined `time` reward structure as:
$$\mathsf{R}^{\mathtt{time}}_{\mathsf{max}} = ?[\mathsf{F}\ \mathit{DoseReady}]$$
Note, that here properties are used to reason about a best- or worst- case scenario, which is typical of the type of query commonly performed business process models at the design phase.

Because model checking is exhaustive and computes exact answers, query values are usually generated PRISM for all states of a model which are explored as part of evaluating a query. Therefore once one query is computed subsequent queries which can be resolved by only querying a subspace of statespace can be rapidly resolved by a simple lookup of the cached results.

4.6 Chapter Summary

This chapter provides a brief overview of the main informatics-based approaches to the formal verification of systems. The choice of model checking for analysis of business processes is justified based on: the lack of a need for a complete specification (properties can be verified individually), the fully automated nature of model checking, powerful design time verification features and its ability to address systems of none-trivial complexity. An overview of the main idea of model checking is given and a survey of the specific quantitative probabilistic model checking tools are presented. The PRISM modelling and property specification languages are described, followed by an analysis of the semantic impact of using PRISM to perform verification of a model.

Chapter 5

Analysing Business Processes

"Computers are useless. They can only give you answers."
*(*Pablo Picasso 1968*)*

Overview

This chapter addresses Objectives 2a, 2b and 2c. An algorithm is developed for the translation of process graphs into models amenable to model checking by means of th Probabilistic Symbolic Model Checker (PRISM) model checker and the algorithm's termination and complexity bounds are accounted for. Pre- and post- processing steps that enhance the precision of the analysis and which allow for bounded rewards values are presented. These allow computation of a broad range of verification and performance properties. The application of the analysis is illustrated using the example introduced in Section 3.4, where a range of properties is verified and provisioning of resources is demonstrated. A natural extension of this verification approach is presented where scheduling is made possible means of exhaustive analysis of possible sequencing of actions in a business process.

The algorithm for conversion of Stochastic Core Business Process Model and Notation (BPMN) models into PRISM code was first sketched in [11], with a refined version appearing in [14], and an improved formulation being made in [12].

5.1 Related work

By employing formal methods to determine quantitative and qualitative properties of BPMN models, this work draws a comparison with a number of other BPMN analysis techniques outlined below. The selection of analyses discussed is not exhaustive, but covers the main approaches which have been widely referenced in the literature.

5.1.1 Functional Analyses

In terms of the analysis of functional qualitative properties a wide range of approaches have been developed for BPMN. These are predominantly focused on the analysis of a limited sets of functional properties, such as proving the absence of deadlocks.

The work of Ouyang et al. [214], [215] is the closest match to the type of translation approach taken in this chapter. Here, translation of BPMN models is done directly into the web-services orientated Business Process Execution Language (BPEL) [211] by means of an algorithm similar to what is presented

in this thesis. However, the approach by Ouyang et al. is intended to support simulation through execution of the BPEL services with all the limitations, detailed in section 4.2, that a simulation approach entails.

Dijkmana et. al. [88] have employed a very similar method to translate BPMN models into Petri-nets, specifically the workflow nets of Aalst [18], which is common to most Petri-net based analysis approaches. Analysis of the generated models is performed by means of the ProM [91] tool which is limited to determining unreachable states (i.e., there are no tasks present in a model which can potentially be executed) and detecting deadlocks. Further, this approach only allows for limited scaling with the authors able to tackle models of up 40 nodes.

The work of Wong and Gibbons [289], employs the Failures-Divergences Refinement (FDR) [60] for the analysis of BPMN. An interpretation of semantics of BPMN are expressed using the Z notation [267] which in turn is translated into Communicating Sequential Processes (CSP) [138] to allow for analysis by means of the FDR model checker. The entire translation is extremely rigorous and includes treatment of the inclusive-OR join construct of BPMN where Wong and Gibbons characterize the inclusive gateway as accepting tokens on some subset of the incoming sequence and then generating tokens on some subset of the outgoing sequence. The use of the FDR model checker imposes the restriction that business processes can only be verified for Computation Tree Logic (CTL) and Linear Temporal Logic (LTL) properties, while quantitative properties can not be examined. This is because FDR is technically a refinement checker, in that it converts two CSP process expressions into Labelled Transition Systems (LTSs), and then determines whether one of the processes is a refinement of the other within some specified semantic model. No support for rewards or probabilistic behaviour is included. Extension of Wong and Gibbons' approach to support these two elements would require the use of different variant of CSP and a different model checker tool, requiring a complete reworking of their approach.

Puhlmann and Weske [233] present the foundations of a tool for static analysis of BPMN process models. This tool, developed as a collection of scripts, relies on a mapping from a subset of BPMN to π-calculus. However, this mapping only covers a small subset of BPMN. Puhlmann and Weske proceed to show that the π-calculus expressions produced by their tool can be used to check the soundness of BPMN models using existing reasoning tools based on the π-calculus, in particular they employ the Mobility Workbench [279], for deciding weak open bisimulation equivalence on π-calculus processes. However experiments, by the authors, show that this approach does not scale beyond relatively small BPMN models of less than 10 nodes [88].

The work of Awad et. al. [36] takes an approach to the verification where a subset of CTL properties of business processes are analysed using the Petri-net model-checker LoLA [260]. This analysis is achieved by converting BPMN models into Petri-nets by means of the previously mentioned method of Dijkmana et. al. [88], fundamentally relying on the workflow-net variant of Petri-nets. A novel inclusion is the development of a visual specification language *BPMN-Q* for the definition of CTL properties of interest. Unfortunately this visual language restricts the possible CTL properties which be expressed limiting analysis to a subset of CTL, with the choice of constructs motivated by a survey by Dwyer et. al. [93] of commonly used patterns. However, a key advantage of this work is that, by means of so called anti-patterns, counterexamples generated by model checking can be employed to provide a complete determination of all points in a model which violate a property of interest and these anti-patterns can be mapped back to the source BPMN model to determine to source or sources of a violation. Finally, it is not clear to what extent the Petri-net model-checker LoLa, while featuring symmetry reduction and partial order reduced by means of stubborn sets, is able to scale to handle large models. Work on the scaling of this approach by Awad et. al. [33] is focused on analysing models of roughly equivalent size, drawn from a repository of multiple separate models, where only subgraphs of models are analysed, which clearly provides a speed up when determining BPMN-Q queries, however in this case mapping results to source BPMN models is not possible. Later work done by Awad with Sakr [35] focuses on the scaling of the BPMN-Q approach when dealing with repositories and not with models of increasing size. Further, stochastic and quantitative extensions are not considered by Awad.

Ly et. al. [179] present another approach to the analysis of BPMN models. Their SeaFlows framework provides a trace-based analysis of business process models for static compliance validation. The set of verification properties which can be determined are restricted to a subset of what can be described using LTL. In earlier work by Ly et al. [180] a visual specification language known as Compliance Rule Graphs is defined, simplifying the specification of these properties. Stochastic and quantitative extensions would not seem to possible to incorporate in this approach.

5.1.2 Non-Functional Analyses

A number of different approaches have been developed to analyse non-functional properties of BPMN models. In particular there has been a focus on determining timing properties of BPMN models. General quantitative analysis has only been identified as being explored by Prandi et. al. [230].

Later work by Wong and Gibbons [290] complements their previous work by developing a CSP based semantics for BPMN. By employing an extension of CSP which accounts for processes *responsiveness*, and an analysis technique that addresses the of timing of processes in such a fashion that their previous FDR based analysis is still possible. This impressive work allows for extensive analysis of BPMN models, but still does not cover arbitrary quantitative properties or stochastic behaviour.

Work by Ling and Schmidt in 2000 [177] present a time interval extension of the workflow nets of van der Aalst [28] in terms of timed Petri-nets. Their extension introduces a notion of safety in time which is analogous to the notion of boundedness in Petri nets [220], using this notion they basically preserve the original soundness theorems of workflows and the liveness theorems of Petri nets in their timed workflow nets. However, their work only provides for placing bounds on the total execution time for a processes and do not allow for analysis of time intervals on a per state basic. Further, treatment of other quantitative properties associated with a business process are only dealt with to a limited degree and due to need to consider completion run times, and may not readily extendible to other quantitative properties of interest.

Additionally Van der Aalst et al. [25] have developed another process execution time prediction method based on process mining. This is done by examining event logs from runs of an existing process to make predictions about the completion time for future runs of the process. This approach could likely be extended to examine other quantitative properties provided log data exists for these properties. However, while this approach has tool support and has been applied to real-life event logs, it does not provide predictive information about expected behaviour of new processes at design time.

Work that combines both timing properties and stochastic behaviour, although aimed strictly at predicting remaining process time, is presented by Rogge-Solti and Weske [247]. They developed a method, using a simulation approach, as opposed to analytical analysis, based on a stochastic Petri-net formalism built upon workflow-nets, that allows for the time taken to perform a task to be given by an arbitrary stochastic distribution. Analysis of these processes is handled by means of Monte Carlo simulation techniques. Their work is focused on the case when a process has already been implemented and data exists describing the execution time of previous executions of a task. Given this an improved prediction is then made for future execution time of this task.

Netjes et. al. have developed an approach [198] to determining the effect of various allocation strategies for resources associated with BPMN processes, based on a simulation approach. In this work each resource is assigned either to a single role or are classed as generic and can be employed by task that requires resources.

Here analysis is performed by means of the simulation in CPN tools [29] to examine whether priority based allocation or random allocation makes optimal use of the resources in question. However, the extension of models with resources is a manual process with manual model modifications required to imposed their depletion semantics, further the results are obtained by simulation as opposed to analytic analysis.

While not directly working with PRISM, Hajo Reijers [244] in his PhD thesis presents two analytical methods for performance evaluation of workflow nets. Here a stochastic variant of workflow nets is studied in which processes have access to an infinite amount of associated resources. Both performance evaluation methods allow for computing throughput time bounds for a general class of workflow nets. Further work focuses on the proper allocation of resources to minimize the throughput time of the workflow. While extensive, this work lacks the development of an end-to-end framework to practical PRISM analysis and the range of quantitative properties which can be studied is limited to throughput time and the effect of specific resource allocation strategies.

5.1.3 Stochastic Analyses

Analysis of BPMN models extended with stochastic properties has seen limited development with only two approaches identified as dealing with general models which exhibit both probabilistic and non-deterministic transitions.

Prandi et. al. [230] have identified PRISM as ideally suited to the analysis of stochastic PRISM business processes. This effort involves conversion of PRISM models into a model expressed in the Calculus for Orchestration of Web Services (COWS) [232], which in turn is converted into a model that can be analysed using PRISM [170]. This approach adds unnecessary complexity in that it is possible to convert the notation of BPMN directly into the BPMN modelling language, and then allow PRISM to impose a semantic interpretation (as detailed in section 4.5.2) without the additional semantic restrictions of going via COWS. Further, the translations PRISM to COWS and in particular from COWS to PRISM is loosely defined and, in the form described by the authors, not amenable to algorithmic translation. This approach, however, does allow the use of rewards. Consequently, the PRISM model checker is potentially able to perform analysis of both quantitative and stochastic properties of a business process. However, details of how such properties will be included in the original BPMN models is not described.

Braghetto et al. [57], [58] provide a mapping from a BPMN Process diagram to a Stochastic Automata Network (SAN), a compositionally built stochastic model. Here probabilities are associated with all exclusive gateways in a fashion similar to Definition 3.6. Furthermore, SAN models, once produced, can be further annotated with rewards although this is a manual process. The main contribution of this work is an algorithm that converts BPMN diagrams to SAN models in the Performance Evaluation of Parallel Programs (PEPS) model checker format. The model checker PEPS [59] is employed to provide numerical evaluation of processes' performance. However, the PEPS tool seems to have seen limited development and does not yet support many techniques reducing the statespaces of SAN models being analysed. Hence the size of models which can be tackled by the tool in a practical setting are limited in size.

In particular, in the area of stochastic business processes and their analysis, only limited results have been achieved. However, addressing stochastic behaviour is essential if business processes are to model real-world behaviour. The analysis approach developed here exploit the tight coupling between the modelling formalism developed in Chapter 3 and the analysis framework developed in this chapter. This is achieved through the structure of Markov Decision Processes (MDPs) which enable an effective representation of stochastic systems combined with nondeterminism.

5.2 Analysis Design

The framework for the analysis of business processes presented in this thesis makes use of the formalised models of business processes defined in Chapter 3. The central goal of this chapter is to present a method to make them amenable to mathematical analysis by means of the PRISM model checker tool as motivated in Chapter 4.

The overall design of the analysis framework developed is shown in Figure 5.1. In line with Objective 2 the goal is to develop an automated approach that is readily usable to business practitioners. Specific details of this architecture are presented in the following sections.

The only inputs required are the specification of a Core BPMN model annotated with probabilities and reward structures where required, and one or more Probabilistic Computation Tree Logic (PCTL) queries (each highlighted in grey). Other than the limited inputs required of a business practitioner, the key strength is the ability of parallelize the model checking of variants of a Core BPMN model. While in the case of checking a single PCTL property only one instance of PRISM

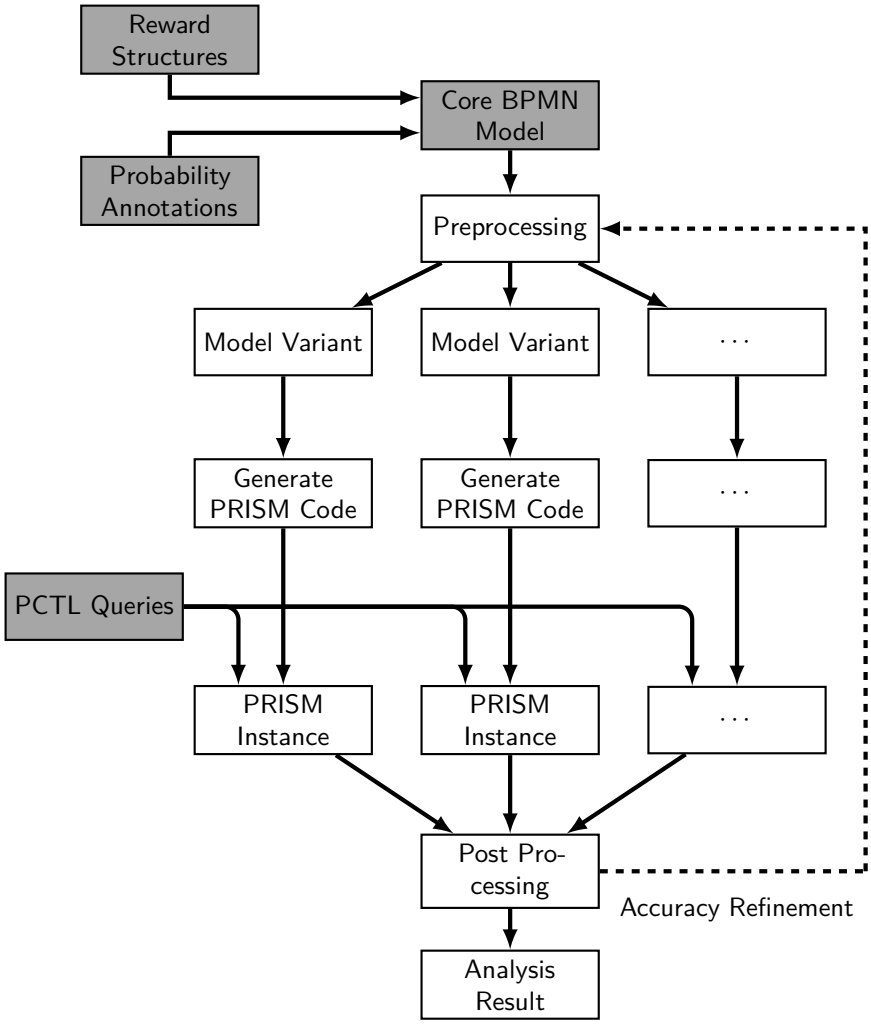


Figure 5.1: *The overall approach taken to automated analysis of business process modelled in Core BPMN models*

is employed, when determining the properties of model for a range of possible resource bounds or when exploring modified Core BPMN models as in the case of the optimisation approach presented in Chapter 8.

The use of PRISM as the underlying model checker allows for highly efficient statespace generation and exploration for Core BPMN models that are of a size typical business for common business process models. Each PRISM execution

allows the efficient exploration of the *entire* state space of a model and enables the calculation of exact probabilities and reward values of properties of a business process, as opposed to the approximate techniques that are typically used by business simulation frameworks.

Here, the application of model checking methods to business processes allows for establishing the certainty that a process exhibits, or does not exhibit, certain properties, such as that it is deadlock free, in line with Objective 2a. Further, the ability to query arbitrary quantitative properties, in line with Objective 2b allows one to perform *provisioning* where the effects of allocating specific quantities of resources to a process can be explored. In all cases queries of the model can account for their stochastic behaviour and determine probabilities in line with Objective 2c.

An implementation of this framework is described in Chapter 9 and an evaluation of its performance on a range of model sizes is detailed in Section 9.3.2.

The mapping of stochastic Core BPMN models to the input format used by the PRISM model tool to describe MDPs fundamentally deals with a translation of a graph based language into a block-structured language used by the PRISM model, in a fashion inspired by the the approach of Ouyang et al. [214], [215]. As such this approach can be adapted to other fundamentally graph based approaches such as Unified Modelling Language (UML) statecharts. In these cases the formalism of process graphs provides the basis upon which a mapping to a stochastic variant of the graph based language can be built and then formalised by means of structural semantic rules, in a style similar to what is done for BPMN in Chapter 3. Common to all of these graph based to block based approaches is addressing parallelism which is the central source of complexity in this approach to mapping models to a format amenable to model checking by PRISM.

5.3 Pre- and Post- Processing

When performing analysis of a model, a number of pre-processing steps are performed to make effective use of bounded rewards, and to ensure that bounds on resource limits lead to a state which captures exhaustion of the resource they model. These pre-processing steps transform a given Core BPMN model into the same model type and the output of pre-processing is still a Core BPMN model.

Further, a combination of Pre- and Post- processing allows for adjusting the size of bins used to express transition rewards modelled as samples from probability distribution using the method of Shimazaki and Shinomoto [262] described in Section 3.2.3.

When implementing this analysis framework, preprocessing is also the point at which to check compliance of a Core BPMN model with the structural semantics described in Section 3.3.2.

5.3.1 Bounded Rewards

Section 3.2.3 introduced the notion of bounded rewards. Recall that rewards are defined for nodes in Definition 3.7 as monotonically increasing structures. Hence, when adding an upper bound to a node reward annotation, the definitions ensure that as a business process is executed, a bounded reward structure will eventually reach its bound. In the preprocessing of bounded rewards, states are added to the source model which capture a reward being exhausted.

At the state (node) n_i at which the reward is incremented and the bound declared, a transition is added to the point of exhaustion of the given reward structure. Control flow at this point is described using PRISM *action guards* which ensure that normal system execution takes place when the bound is not met. A transition to the deadlocking reward exhaustion state is taken in the case when a reward meets or exceeds a bound. This process is illustrated in Figure 5.2.

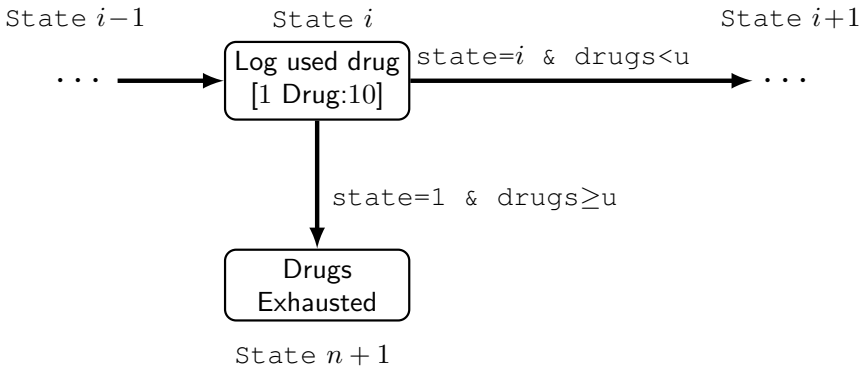


Figure 5.2: *Reward exhaustion illustration drawn from the example in Section 3.4.*

In Figure 5.2, it is assumed that all states of a source model have been enumerated and n states have been found. An additional $n+1$ exhaustion state (recording that drugs have been exhausted in Figure 5.2) is added to the model. When translating a model into the PRISM modelling language, a new PRISM variable v_i which ranges from zero to the bound u is added to the module. For example, drugs in the illustration in Figure 5.2. Transitions from state n_i are defined using the following function:

Definition 5.1 (Reward Exhaustion State Injection Function)
For a node $n \in \mathbf{N}$ in a process graph and a reward bound $u \in \mathbb{R}_{\geq 0}$ the partial function $\mathcal{W} : \mathbf{N} \times \mathbb{R}_{\geq 0} \rightarrow (\mathbf{N} \times \mathbf{N})$ adds a reward-exhaustion transition to a progress graph as follows:

$$\mathcal{W}(n,u) = \begin{cases} n_i \boldsymbol{\mathcal{F}} n_{i+1} & \textit{with guard } \texttt{state} = i \,\&\, \texttt{reward} < u \\ n_i \boldsymbol{\mathcal{F}} n_{n+1} & \textit{with guard } \texttt{state} = i \,\&\, \texttt{reward} >= u \end{cases} \tag{5.1}$$

When the pre-processed BPMN model is converted into the PRISM modelling language the added guards are copied verbatim into the resulting PRISM code.

Many business processes make use of various limited resources such as stock levels, or time. The accurate provisioning of these resources, in the form of *quantitative service analysis*, can increase their safe and efficient execution. It is frequently desirable to perform *provisioning*; where a model is analysed for a range of bound values to determine an appropriate amount of a resource to allocate to a business process so that it will operate within specific bounds. This type of analysis is achieved in the framework at this preprocessing stage. When a defined reward bound is encountered in the process of generating PRISM code, separate models are generated for the range of values between 0 and the bound u for the given reward. Here it is convenient to define a step size of different reward bounds to test in the range 0 to u.

In terms of the analysis, the generation of a range of models for real world examples can be large. However, the structure of the statespace enjoys a large degree of symmetry, and symmetry reduction optimisations [167] implemented in PRISM allow for a significant reduction in the complexity of each individual model. An example of the generated state space for the BPMN example from Section 3.4 is given in Figure 5.3.

In post-processing, the results for separate model checking runs of queries of interest with different bound sizes are gathered together. It can be convenient to impose a time bound on individual model checking runs and terminate those that are taking too long, and then interpolate between obtained data points.

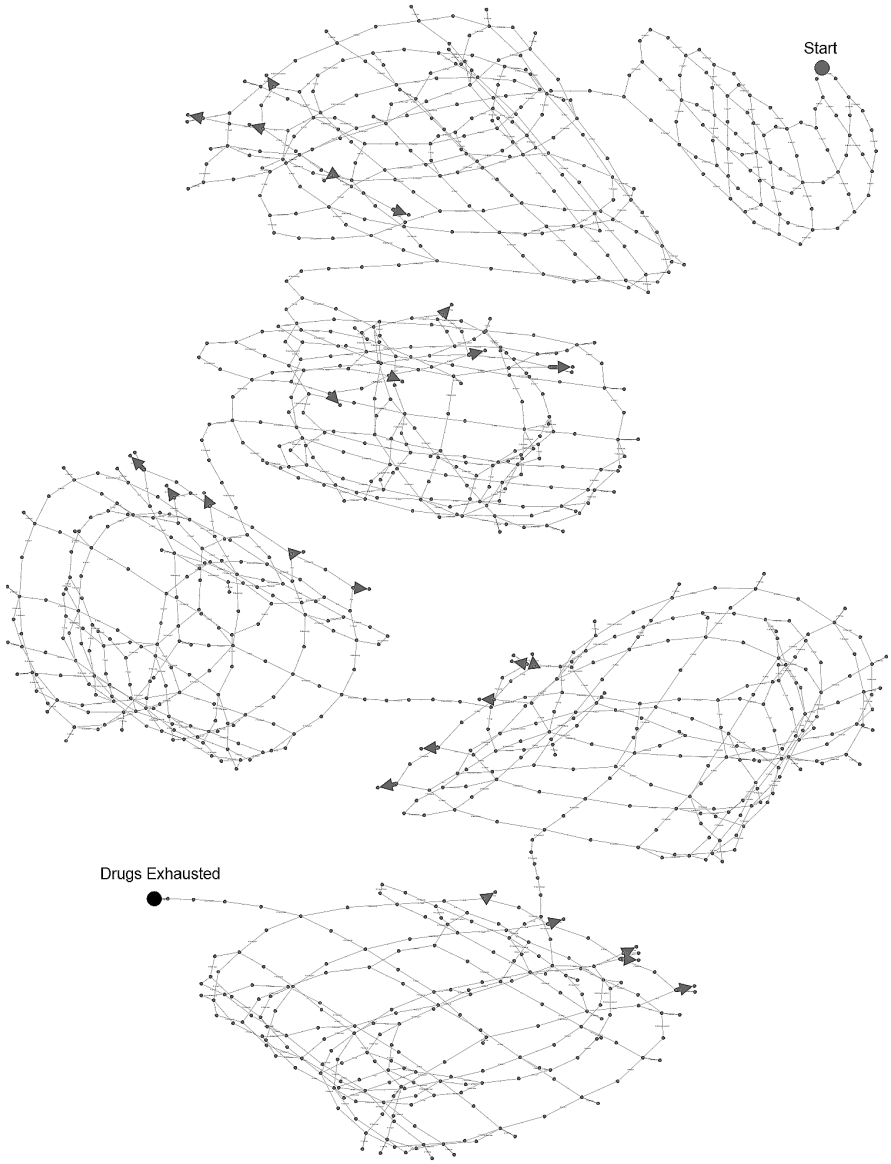


Figure 5.3: *Statespace for the example from Section 3.4 with drug stock limited to* 5*, the triangles indicate states in which the patient is sent home (367 states, 796 transitions).*

Although these interpolated values may not capture the actual behaviour of the system and in an implementation should be flagged as approximate values. This

approach to implementation has utility as the benefits in terms of computation speed can be useful when a quick estimate of the behaviour of a system is required. A more comprehensive analysis with a larger, or entirely omitted bound, can then be performed once a final complete analysis is required.

In the case when the PCTL query analyses a property which is not affected by the reward bound, or where a model is defined in such a fashion that no possible execution path exists in which the bounded reward is incremented, checking a model for a range of reward bounds will produce the same answer to the PCTL query for the full range of values. In the case of exploring probabilities, a query with a fixed probability, or, a query over a unbounded reward structure, a fixed value (including possibly ∞), will be returned for all reward bounds explored, e.g. querying how long it takes to exhaust a supply of drugs which is never used will return ∞ for all possible sizes of a drug store.

The performance of checking bounded rewards is explored in Section 9.3.1.

5.3.2 Transition Rewards Sampled from a Distribution

Section 3.2.3 introduced the notion that *transition rewards* can be used to capture samples from a probability distribution. When performing analysis of such structures it can be beneficial to discretize the distribution's probability density function by employing the technique of Shimazaki and Shinomoto [262] for choosing an optimal bin sizes for taking a chosen number n of samples that correspond to intervals from the space of possible reward values, i.e. $p_n = Pr[a_n < X < b_n]$ where a_n and b_n are the bounds of the interval n. For example, if the reward structure models the time between two steps in a process and is normally distributed, a number of samples can be constructed which correspond to some fraction of the distribution's standard deviation.

For a fixed convergence value of PRISM's PCTL checking algorithms, it can be useful to refine the number of samples made from a distribution so as to improve the accuracy of the results obtained. Here, a fixed number of samples are made, and the values of queries of interest are then calculated. A second run is then performed for an increased number of samples from a distribution. If the result changes beyond a chosen parameter δ, this new model is used as the basis for the next iteration of model checking until increasing the number of samples taken does not produce a change greater than δ of the result of a query. The ability in the design of the analysis to perform multiple model checking runs allows the number of samples taken from each distribution to be explored in an efficient manner.

5.4 Translation to PRISM Models

The central idea of the translation algorithm is to identify sub-processes of the source BPMN model, and then map these to modules of PRISM code so that the encoding is compositional and will not impose further semantic interpretation on a model than was originally defined. This mapping, which focuses on the control flow structure of the model, involves the challenge of mapping a graph based language to a block based language. In the case when BPEL is the target language, a technique has been developed by Ouyang et. al. [214], [215]. The overall approach of their work forms the basis for the developments presented here.

For reasons of simplicity, only the translation of a single pool of a BPMN model is considered here. Further pools can be translated by running the algorithm again for any additional pools, although with a new set of module and variable names for each iteration.

Translation is performed by means of two algorithms. The translation starts by employing Algorithm 1 to build lists of pairs of linked nodes in the source model at each level of nested parallel fork gateways. These pairs are stored in an array $MAP[d]$ which records which pairs of nodes are present at each level of fork gateway depth d of the BPD pool. The corresponding fork and merge gateways present at a given level are also present in both the level above and below the fork depth, where in the containing level a fork gateway is linked directly to a merge gateway.

Algorithm 1 takes as input a Core BPMN and only operates on the nodes of the source model and not on its implied state space. Algorithm 1 proceeds as follows: in lines 1 and 2, a counter d which records the current nesting depth is initialised to zero (the starting nesting depth) and the start event is pushed to a stack $STATES$ of nodes to be processed. Lines 3 to 15 are a *while* loop during which the nodes of a BPMN model are added to the array $MAP[d]$. Specifically, a state n is popped off the stack (line 4) and added to a set $VISITED$ of visited state (line 5). In line 6 the state to process is checked to see *if* it is a task, start event, decision gateway or parallel join gateway. In the case when n is one of these nodes, all nodes m in the set of successor nodes $\mathsf{out}(n)$ (where out is as defined in Definition 3.3) are added as pairs (n, m) to the array of nodes in $MAP[d]$ at the current nesting depth d (line 8). Each m which is not already in the set $VISITED$ is added to the $VISITED$ set (line 9) and these nodes (m) are pushed onto the stack $STATES$ of states to be explored (line 10). Next, in line 11, a new stack of nodes $SUBPROCESS$ is initialised. Then *if* node n is a parallel forking gateway and not already in $VISITED$ (line 12) then a depth first search is performed to find the matching parallel join gateway g to n (line

Algorithm 1: Branch depth state grouping

Input: A well-formed $BPD = (\mathbf{N}, \boldsymbol{\mathcal{F}}, \mathbf{P}, \mathsf{pool}, \mathbf{L}, \mathsf{lab})$
Output: $MAP[d]$

```
1  d ← 0
2  STATES.push(e_s)
3  while STATES ≠ ∅ do
4      n ← STATES.pop()
5      VISITED ← n
6      if (n ∈ T ∪ E^S ∪ G^D ∪ G^J) then
7          forall the m ∈ out(n) do
8              MAP[d] ← MAP[d] ∪ (n, m)
9              if (m ∉ VISITED) then
10                 STATES.push(m)
11     SUBPROCESS ← ∅
12     if (n ∈ G^F ∧ n ∉ VISITED then
13         Depth first search for n's matching g ∈ G^J
14         MAP[d] ← MAP[d] ∪ (n, g)
15         SUBPROCESS ← path(n, g)
16 if SUBPROCESS ≠ ∅ then
17     d ← d + 1
18     STATES ← SUBPROCESS
19     Goto line 3
```

13). Following this in line 14 the pair (n, g) is added to $MAP[d]$ and in line 15 the nodes in the $\mathsf{path}(n, g)$ (see Definition 3.11) are added to the $SUBPROCESS$ stack. *If* the $SUBPROCESS$ stack is not empty then the depth counter is incremented and (line 17) the $SUBPROCESS$ stack replaces the $STATES$ stack (line 18)and in line 19 execution jumps to 3.

Algorithm 1 has an upper complexity bound of $O(n^2)$, where n is the number of nodes in the source BPMN model. This upper bound corresponds to the pathological case when a BPMN model consists of nothing but nested forking gateways, within the limitations of the structural soundness conditions given.

With $MAP[d]$ defined, the actual translation of a Core BPMN BPD to one, or more, PRISM modules described in the PRISM modelling language is performed by Algorithm 2.

Algorithm 2: PRISM Module mapping

Input: $MAP[]$ an array of sets of pairs of nodes at level of branching depth
Output: PRISM Module(s)

```
1  for (d ← 0, MAP.length(), d++) do
2      Emit( module M_d )
3      forall the (x, y) ∈ MAP[d] do
4          N ←< x, y | x ∉ N ∨ y ∉ N >
5      Emit( s_ d :[0 .. N.length() ] init 0 )
6      l ← 0
7      forall the (x, y) ∈ MAP[d] do
8          if x ∈ T ∪ E^S ∪ E^E then
9              Emit( []: s_d=N.pos(x)→1:s_d=N.pos(y) )
10         if x ∈ G^D then
11             i ← 0
12             forall the (z ∈ xSz) and (l∈ lab(z)) do
13                 up_i ← P(((x, z), l) :s_d=N.pos(z)
14                 i ← i + 1
15             forall the (l∈ lab(z)) do
16                 Emit( [l]: s_d=N.pos(x)→ )
17                 for (j, i, j++) do
18                     Emit( +up_j )
19         if x ∈ G^F then
20             Emit( [l_l]:s_d=N.pos(x)→1:s_d=N.pos(y) )
21             l ← l + 1
22     Emit( endmodule )
```

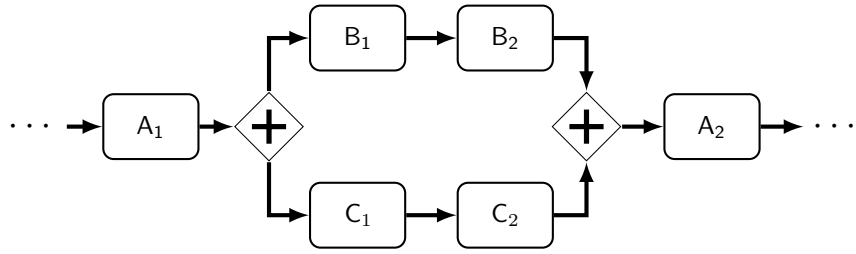
Algorithm 2 takes as input the $MAP[d]$ the array of pairs of nodes that are present at each level of fork gateway depth d and outputs **module** blocks. Algorithm 2 loops through the elements of $MAP[d]$ by depth d. Each iteration of the loop emits a PRISM module and in line 2 the module header is output, denoted here by an "Emit" command. Next, in lines 3 and 4 an ordered list N of the elements of $MAP[d]$ for the current loop iteration's value of d. Line 5 emits a PRISM state variable which ranges from 0 to the size of list N. In line 6 a label counter is initialised to 0. Lines 7 to 20 are a loop through pairs of nodes (x, y) from $MAP[d]$ and emit appropriate PRISM language commands for each type of node. If the first node x, in the pair (x, y), is a task, start- or end- node (line 8) then, in line 9, a PRISM a command is emitted which captures the transition of state variable s_d, with probability 1, from the position of x in N to the position of y in N. If node x is decision gateway (line 10), a counter i is initialised to 0 then,

in line 13, all of the labelled transitions from x to nodes z produce updates to states z with a transition probability dictated by the function $\mathcal{P}$ (Definition 3.14). In lines 15 to 18 the separate update commands for a decision gateway are combined into a single command. By construction of the first element of pairs in $MAP[d]$ are never merging gateways, so the final case of a parallel forking gateway is dealt with in line 19. In this case when x is a parallel forking gateway the node y, in the pair (x, y), with be the corresponding merge gateway at the same nesting depth as the original fork. In line 20 a PRISM command is emitted which is guarded by a new label $\texttt{l}_l$ encoding a transition to the corresponding merge gateway y, following this, in line 21, the label counter is incremented. Finally, in line 22 the end of PRISM module is declared.

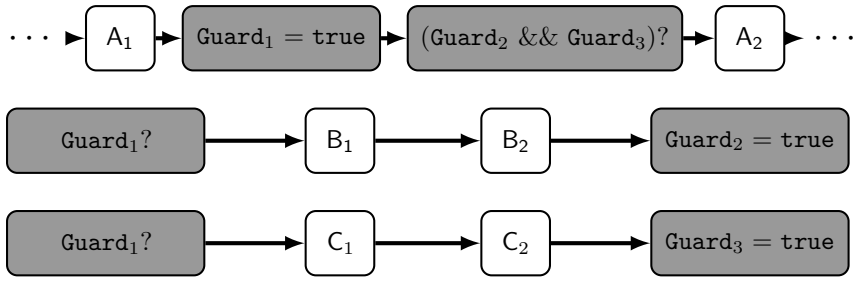
Algorithm 2 simply loops through elements of $MAP[]$ and hence has upper a complexity bound of $O(n)$, where n is the number of pairs in $MAP[]$.

The combined effect of Algorithm 1 and Algorithm 2 is to traverse a Business Process Diagram (BPD) and output PRISM code, arranged within **`module`** blocks, for each BPD component that is encountered. Basic components (start, end, tasks) are dealt with by outputting a transition to the unique next state. Decision gateways employ the associated probabilities to define multiple transitions to new states. The more complex case of parallel branch gateways makes use of recursion, the main module will describe a transition from the branch point to the merge point with guards placed on this transition that the generated sub-processes describing the parallel section must have completed. Then the same Algorithm 2 is applied to the sub-processes with guards generated so that they can only start when the main module has reached the branch point. Finally, the sub-processes are reset for a possible second iteration by requiring of the merge gateway state, by means of guarded commands, that all sub-processes have returned to their start state. Producing an interleaving semantics for concurrent processes is described in Section 4.5.2.

This synchronisation between modules is fundamentally accounted for by generating appropriately named PRISM *actions* that enforce synchronisation between modules. An illustration of this process for a simple pair of parallel fork and merge gateways is shown in Figure 5.4(a), and the resulting three guarded PRISM modules are shown in Figure 5.4(b). It should be noted that while this synchronisation between sub modules is performed by means of PRISM action labels, these are ultimately converted, due to the semantics imposed by PRISM (See Section 4.5.2), into additional system module variables which impose synchronisation be means of dedicated state variables.



(a) Parallel BPMN Process



(b) PRISM Modules

Figure 5.4: *Algorithm 2's approach to handling parallelism in a Core BPMN model. Tasks* $\mathsf{B}_1, \mathsf{B}_2, \mathsf{C}_1, \mathsf{C}_2$ *will be executed after* A_1 *in all possible interleavings and then followed by the execution of* A_2*.*

Message passing between pools is addressed in a similar fashion to dealing with sequence flows in a single pool. Message passing between models is addressed once each pool has been translated. Appropriate synchronising PRISM actions between them are generated in the fashion shown in Figure 5.4(a) and Figure 5.4(b) by means of appropriate PRISM *synchronising actions*.

In a practical implementation of Algorithm 1 and Algorithm 2 it is beneficial, in terms of analysis, to include the names of tasks and pools in the resulting PRISM code. Here it should be noted that labels are chosen so as not to conflict with previously used labels.

The treatment of rewards has been omitted from Algorithm 2. The treatment of rewards is addressed by simply mapping each reward structure to a PRISM *reward module* with the rewards identified by a state for *node rewards* and by a *synchronisation action* for *flow rewards* in line with Definitions 3.7 and 3.8.

5.5 Analysis Example

To illustrate the application of the developments presented in this chapter, consider the example from Section 3.4 of a doctor treating a patient by means of drugs. Despite the example being conceptually simple, the resulting state space, shown in Figure 5.5, exhibits greater complexity with 79 states and 172 transitions. Even in this simple example, the number of possible specific situations in the business process that can arise, corresponding to individual states in the state space, is non-trivial.

Nevertheless, the analysis described in this chapter can be employed to verify a range of key properties of this business process. A simple qualitative property of the example would be to verify that the a patient is eventually sent home. This is captured in queries eqs. (5.2) and (5.3) using explicitly quantified queries over the P operator:

$$\mathsf{P}_{\mathsf{max}} \geq 1.0[\mathsf{F}\ \mathit{SendHome}] = \text{true} \tag{5.2}$$

$$\mathsf{P}_{\mathsf{min}} \geq 1.0[\mathsf{F}\ \mathit{SendHome}] = \text{false} \tag{5.3}$$

A flaw in the design of this business process is revealed by query eq. (5.3) returning a bound on the probability is not greater than or equal to 1. Hence it is possible that after a patient receives the first dose of drug, the treatment is not effective and the patient requires a second dose. However, in this case it is possible that meanwhile the pharmacist has made a non-deterministic *leave* decision and is not available to prepare the second dose. Consequently the patient will not receive their, required, second dose. Note, how this statespace can be matched to the trace semantics given for Core BPMN models, although a number of intermediate states are added by the translation to PRISM the flaw is preserved in the statespace of PRISM model. This fault can be corrected if the pharmacist process is restricted so that he may not leave the hospital unless a patient has been successfully treated, and the probability that a patient will eventually be sent home will then have an upper and lower bound of 1.

While this example is of a modest size, it demonstrates the ability of a model checking based framework to determine key properties of stochastic business processes. The use of PCTL to express properties of interest allows for a rich logic in which business process can be analysed for both by qualitative and quantitative properties in line with Objectives 2a, 2b and 2c. A practical implementation and the scalability of this approach will be examined in chapter 9.

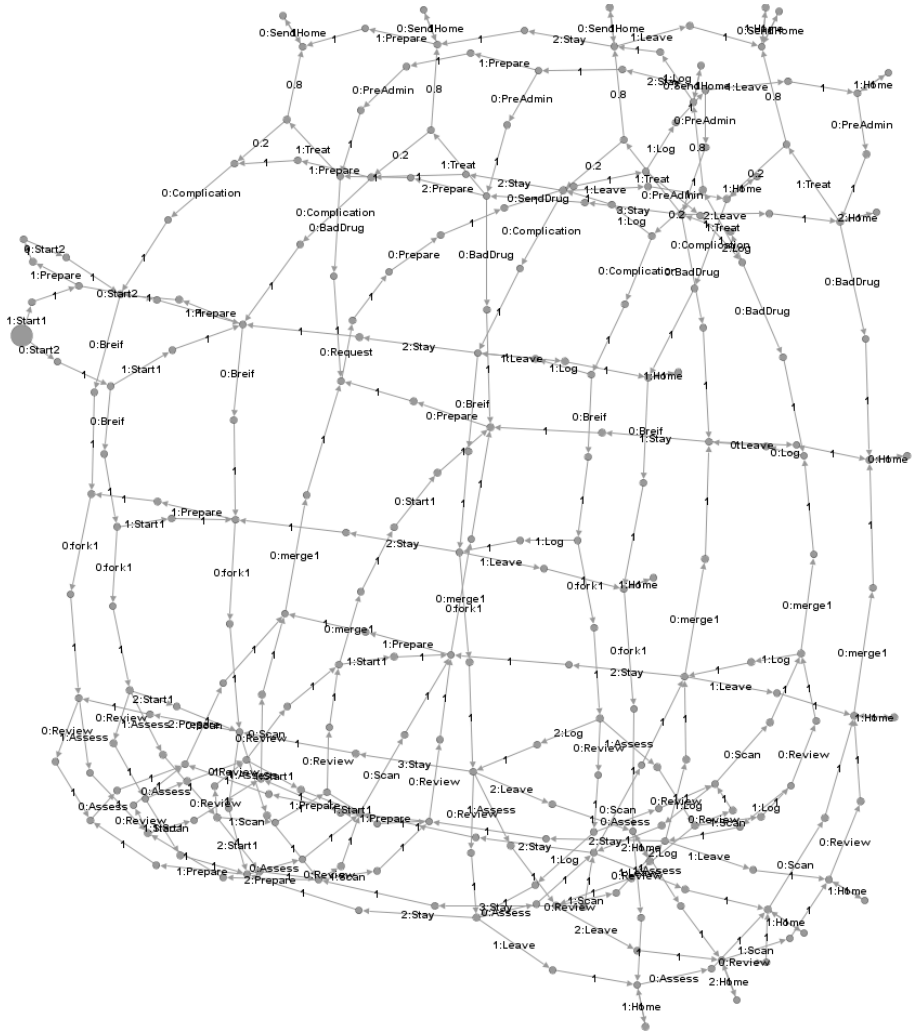


Figure 5.5: *PRISM generated statespace for the example from Section 3.4 with an unbound drug supply (79 states, 172 transitions).*

For the rest of this example a version where this fault has been corrected is considered. Correction can be performed by disallowing *leave* decision, for example by setting its associated probability to 0.

A common quantitative property that would be of interest to the designer of such a process would be the mean time to complete treating a patient. For the corrected process, this can be formulated using the R operator to determine the lower and upper bounds, under resolution of non-determinism, in queries eqs. (5.4) and (5.5):

$$\mathsf{R}^{\mathtt{time}}_{\mathsf{min}} = ?[\mathsf{F}\ \mathit{Done}] = 44.99989 \text{ min} \tag{5.4}$$

$$\mathsf{R}^{\mathtt{time}}_{\mathsf{max}} = ?[\mathsf{F}\ \mathit{Done}] = 54.99989 \text{ min} \tag{5.5}$$

Note in the results produced by queries in eqs. (5.4) and (5.5) are given to 5 significant digits, the precision of these results flows directly from the original model (presented in Section 3.4) and highlight a key strength of model checking in that the entire statespace is examined and hence exact results produced are able to be produced. Although as with all model based techniques the accuracy of the original model dictates the quality of results obtained.

The analysis methods presented in this chapter can be employed in a more sophisticated fashion to help determine the provision of resources consumed as part of executing a business process when the process is designed. In the example key factors in ensuring safety would be the determination of the amount of drug to stock and the minimum time before its exhaustion. Thus there would be a low probability of it being exhausted during the patient's treatment. Formally, these properties can be expressed using the following queries for, respectively, the probabilities queried by eqs. (5.6) and (5.7) and minimum time needed to reach this state queried by eq. (5.8):

$$\mathsf{P}_{\mathsf{max}} = ?[\mathsf{F}\ \mathit{NoDrug}] \tag{5.6}$$

$$\mathsf{P}_{\mathsf{min}} = ?[\mathsf{F}\ \mathit{NoDrug}] \tag{5.7}$$

$$\mathsf{R}_{\mathsf{min}} = ?[\mathsf{F}\ \mathit{NoDrug}] \tag{5.8}$$

This can be calculated by making use of the ability of the analysis framework to generate a range of models for a bounded reward. In this case, the rewards modelling the size of the drug stock in the model are bound to a fixed limit l, and then calculating the values of queries in eqs. (5.6) and (5.7) for a range of model variants with values of l from 0 (no store of drug) to 10 units of drug. The results are shown in Figure 5.6.

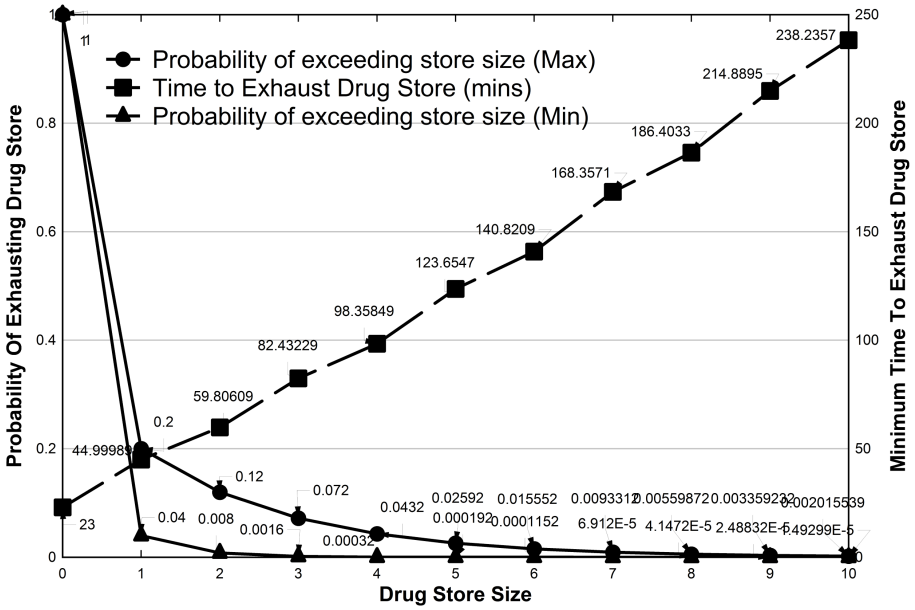


Figure 5.6: *The upper and lower bounds on the probability of exhausting drug stock and minimum time taken to do so for the example from Section 3.4 for a range of drug stores from 0 to 10.*

Figure 5.6 presents an analysis of the effect of allocating various amounts of resources to a business process and provides an assessment of both the probability bounds that a given resource level will lead to resource exhaustion, but also the non-functional property of the expected amount of time taken to reach such states. In various jurisdictions, this type of provisioning analysis is frequently part of regulatory requirements for the introduction of new medical technologies and the ability to perform this in an exact and automated fashion has been of considerable value to the project's industrial partner.

5.6 Chapter Summary

This chapter presents the chosen method for analysis of business processes. The PRISM modelling and property specification languages are described and an algorithmic translation is presented for the conversion of Core BPMN models of business processes into the PRISM modelling language.

By adding bounds to reward structures it is shown how provision of limited resources can be performed, along with verification of both functional and non-functional properties of business processes. An example is given of this analysis and the nature of the statespaces involved in this analysis are discussed.

Building upon the model checking approach to the analysis of business processes, this chapter presents a method to determine an optimal schedule (defined as a choice of actions to be performed in a business process so as to optimise properties of interest) for the execution of a business process. The schedule is created by means of the generation of adversaries which resolve points of non-determinism in a business process such that specific PCTL queries obtain their minimum or maximum value. As such the schedule generated can optimise reward values while observing constraints which encode any required execution properties.

Chapter 6

Schedule Generation

"The only reason for time is so that everything doesn't happen at once" (Albert Einstein)

Overview

This chapter demonstrates how the analysis can be used to synthesise a schedule for the execution of a business process, subject to both functional and non-functional requirements, which addresses Objective 3a.

The analysis technique to resolve scheduling problems were first presented in [12]. Aspects of this work were also presented in [3], [8], [16].

6.1 Schedule Generation

When designing business processes there is a need to combine the verification of safety properties with a specific performance profile. To achieve these goals, business processes often require sophisticated execution schedules especially when the system involves stochastic elements. Being able to synthesize a schedule for the optimal execution of such systems early in their design phase allows for accurate determination of how the system will be employed, in the form of the sequence of actions performed at points of non-determinism in the business process. This holds the potential for the early identification and exclusion of inefficient designs, and suggests which patterns of execution are most likely and should be focus of testing later in the business processes development. A direct application of the analysis technique presented in this chapter is the determination of such schedules, in line with the requirement of Objective 3a.

In a business context, a schedule is defined by Kotter [161] as: "*A decision-making function that plays an important role in most manufacturing and service industries and often allows an organization to operate with a minimum of resources. Scheduling is applied in procurement and production, in transportation and distribution, and in information processing and communication. In manufacturing, the scheduling function coordinates the flow of parts and products through the system, and balances the workload on machines and personnel, departments, and the entire plant.*"

Within this thesis a schedule will be defined as the *sequence of actions to be taken from an initial state in the system to obtain the optimum of one or more quantitative properties associated with a model.*

Model checkers have been used extensively to solve scheduling problems where the basic common idea is to reformulate the static scheduling problem as a reachability problem that can be subjected to model checking. A classic example

of this approach is that of Ruys [256] who presents a method for determining optimal schedules using the SPIN model checker [140]. Here costs are associated with each step in the model and Linear Temporal Logic (LTL) formulas are used to express possible paths through the model. A branch and bound approach [172] is then employed to eliminate paths which exceed an already determined optimal solution or which feature an invalid (deadlocking) solution. This approach is effective in rapidly converging on an optimal schedule and can be extended to account for costs, such as in the work of Behrmann et. al. [44] where priced timed automata are analysed using the the UPPAAL model checker [43] to allow for optimal scheduling over a number of monotonically increasing quantities associated with a model.

While the traditional approach to employing model checking to solve scheduling problems is mostly concerned with resolving all points of nondeterminism so as to minimise or maximise one or more values of interest. However, the addition of stochastic behaviour in a system of interest significantly complicates the application of many traditional scheduling approaches. In this case a schedule is potentially associated with many different executions, which must be all taken into account in order when determining the effect of a specific sequence of actions on the overall goal. Conceptually similar to the approach taken here is the work of Giunchiglia and Traverso [115], where planning problems are seen as including non-determinism not under the control of a planner, and a similar approach to model checking business processes is developed here.

Within the analysis framework presented here, schedule generation is simply another automatic analysis that can be performed once a business process has been modelled. The approach taken to schedule generation is shown in Figure 6.1, where the only additional information needed to generate a strategy for a business process is a scheduling goal and a set of schedulable actions. The result of schedule generation is a Discrete-time Markov Chain (DTMC) which encodes the specific sequence of actions required to best approach the scheduling goal. This DTMC is labelled so as to record the sequence of actions that a schedule encodes and this can be mapped back to the original Core Business Process Model and Notation (BPMN) model so as highlight the specific choices needed to execute the optimal schedule.

Central to the approach taken here to schedule generation is to exploit the generation of *adversaries* (see Appendix A.6), also known as strategies, inherent in Markov Decision Process (MDP) model checking. The generation of *optimal adversaries* in Probabilistic Symbolic Model Checker (PRISM) is determined as part of computing a Probabilistic Computation Tree Logic (PCTL) query for model and capture the specific choice of actions, resolutions to points of nondeterminism, which ensure that a probability of reward query are maximised or minimised. A novel feature, and recently implemented feature in PRISM, is

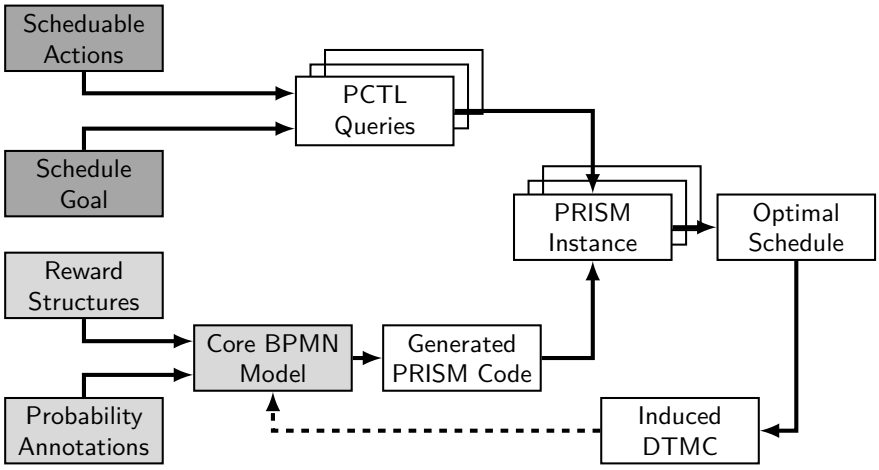


Figure 6.1: *Overview of the Core BPMN analysis framework approach to schedule generation (grey boxes mark inputs needed, with dark grey being the schedule goals and schedulable actions).*

that multi-objective properties can defined for adversary generation [104], [105]. This allows a combination of reward and probability queries to be combined and a single adversary generated which optimises all of these values. It should be noted that PCTL queries can also encode required safety properties such that a generated schedule will optimise the scheduling goals while excluding schedules which violate a safety requirement.

6.1.1 Schedule Specification

Within this thesis a schedule is defined as the *sequence of actions to be taken from an initial state in the system to obtain the optimum of one or more quantitative properties associated with a model.* This can be combined with a set of constraints on the process. These constraints would typically capture safety properties, but can be freely defined using PCTL to express properties that must hold when executing the strategy.

For example suppose one was considering a medical robot capable, by means of making a non-deterministic choice, of producing either drug A, B or C. In the case when three doses of drug A, two doses of drug B and one dose of drug C are required. The possibilities for sequencing these actions are illustrated in in Figure 6.2, where each possible schedule is a path from the root to a terminal node.

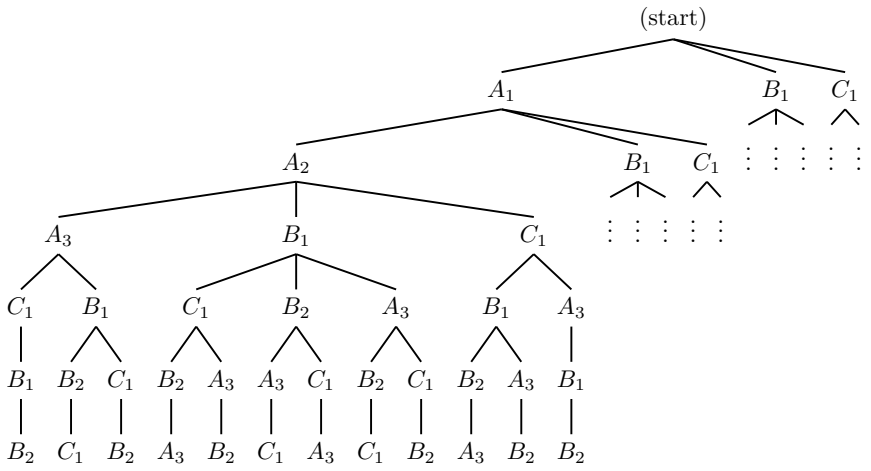


Figure 6.2: *Illustration of the possible scheduling possibilities for a combination of* 3 *dose of drug A,* 2 *doses of drug B and* 1 *doses of drug C.*

Determining an adversary, also known as a *strategy*, for a model with non-deterministic choices as shown in Figure 6.2 determines the effect these possible sequences of actions will have on the reward and probability values encoded in a PCTL query of interest. As PRISM quantifies over all possible adversaries, i.e. all possible resolutions of nondeterminism in the model it is ensured that there exists within these possible resolutions a set of resolutions for which a models associated reward and probability values which take on minimum or maximum values. Given a desired set of scheduling goals, solving a scheduling problem by means of model checking involves determining which schedule optimises the chosen scheduling goals.

Adversary properties for a single objective are specified in the standard fashion for PRISM property queries described in section 4.5.3. Multi-objective adversary properties are specified in PRISM 4.1 by means of the `multi(...)` keyword. This keyword allows for a comma separated list of separate queries to be defined and may only be employed when specifying an adversary.

Queries may seek to determine the an adversary, e.g. in eq. (6.1), given an MDP which adversary comes closest to achieving that with a probability of less than 0.3 a *Load* state is reached and that with a probability less than 0.1 an *Error* state is reached.

$$\texttt{multi}(\mathsf{P}{<}\,0.3[\mathsf{F}\ \mathit{Load}], \mathsf{P}{<}\,0.1[\mathsf{F}\ \mathit{Error}]) \tag{6.1}$$

If a multi-objective property contains a single unbound ? objective then an adversary is determined which , e.g. in eq. (6.2), given an MDP which adversary achieves the minimum possible probability of reaching a *Load* state, for which the probability of reaching *Error* is less than 0.1.

$$\texttt{multi}(\mathsf{P}_{\mathsf{min}}{=}?[\mathsf{F}\ \mathit{Load}], \mathsf{P}{<}\ 0.1[\mathsf{F}\ \mathit{Error}]) \tag{6.2}$$

Finally a C operator is allowed in adversary generation which calculates the total cumulative value of a reward structure in a multi-objective property query. This value is the total of the reward value accumulated along all paths as opposed to simply the a specific path of the adversary as is the case with the F operator. For example eq. (6.3) expresses: which adversary ensures that the expected cumulative value of the reward structure *time* is minimised while ensuring that the expected cumulative value of reward structure *energy* is below 7.2.

$$\texttt{multi}(\mathsf{R}_{\mathsf{min}}(\mathit{time}){=}?[\mathsf{C}], \mathsf{R}_{\mathsf{min}}(\mathit{energy}){<}\ 7.2[\mathsf{C}]) \tag{6.3}$$

Further, possibilities are for specification of multiple adversaries along with an extensive description of the computation of adversaries is available in [105].

6.1.2 Schedule Generation

The adversary generation feature in PRISM produces an induced DTMC over an MDP which optimises the properties of interest by appropriate resolution of points of non-determinism in a model. This induced DTMC that equates to evaluating the best- or worst-case choice of actions at all decision points that satisfy a chosen PCTL constraint. Having produced such a DTMC the sequence of actions present in the DTMC record the schedule and can readily be mapped back to the source MDP to highlight the optimal schedule, this is described in Section 9.3.3. In practice, the use of rewards allow for the determination of ideal schedules with regard to resources consumed, such as, time or energy used, and whereas multiple schedules may exist, quantitative MDP model checking methods allow for the selection of one of the schedules which optimise rewards of interest. At present internal optimisations within PRISM allow only one of the optimal schedules to be returned.

The specific query to be produced to generate the possible schedules for a Core BPMN process involve defining a multiset **A** of actions which must be performed as part of the schedule, with multiple occurrences of the same element encoding that the action must be performed multiple times. Combined with a possibly empty PRISM PCTL query **C** encoding constraints on the strategy. Given **A** and **C**, the schedule generation query is constructed using Algorithm 3.

Algorithm 3: PCTL scheduling query generation.

Input: A set $\mathbf{A}$ of actions and a set $\mathbf{C}$ of constraints.
Output: A PCTL query Q.

```
1  forall the a ∈ A do
2      m ← v(a)    // where v(x) determines the multiplicity of x
3      if m > 1 then
4          for i ← 0 to m do
5              Q ← Q | a U          // where | denotes concatenation
6      else
7          Q ← Q | F a
8      Q ← Q | ∧
9  forall the c ∈ C do
10     Q ← Q | ∧ c
11 return Q
```

Algorithm 3 simply produces a PCTL query Q. When constructing Q each element $a \in \mathbf{A}$ is combined using the PCTL *until* operators, if more than one occurrence of a is present in A, and using the PCTL *finally* operator in the case when a only occurs once. For each a each of the sub-queries are combined using conjunction. Finally, each of the constraints $c \in \mathbf{C}$ are combined with Q by means of conjunction. Note that Algorithm 3 omits the addition of square and rounded brackets need to build PRISM queries and that in line 5 an additional U must be omitted on the final run of the *for* loop.

The complexity of Algorithm 3 is $O(nm)$ where n is the cardinality of $\mathbf{A}$ and m is the is the cardinality of $\mathbf{C}$.

Note that scheduling involves resolving all point of nondeterminism so actions which could potentially be scheduled but which are not included in the set $\mathbf{A}$, i.e. $\mathbf{A}^{\complement}$, may, or may not, be included in the schedule, at any point, depending on what produces the optimal adversary in terms of the scheduling constraints constraints.

Having constructed Q a PRISM adversary generation query can be constructed as follows:

$$\texttt{multi}(\, P_1[Q], P_2[Q], \cdots, P_m[Q])$$

where $P_1, P_2, \cdots, P_m$ are the PCTL reward or probability operators for which the adversary is to be optimised with respect to.

In the case when three doses of drug A, two doses of drug B and one dose of drug C are required, i.e. $\mathbf{A} = \{a \to 3, b \to 2, C \to 1\}$, combined with a safety constraint $\mathsf{F}\ \mathit{Fail}\ \mathsf{X}\ \mathit{Reset}$, the combined query to minimise execution time and cost, would take the form:

$$\texttt{multi}(\quad \mathsf{R}_{\mathsf{min}}(\mathit{time})=?[(a\ \mathsf{U}\ a\ \mathsf{U}\ a) \wedge (b\ \mathsf{U}\ b) \wedge (\mathsf{F}\ c) \wedge (\mathsf{F}\ \mathit{Fail}\ \mathsf{X}\ \mathit{Reset})], \quad (6.4)$$

$$\mathsf{R}_{\mathsf{min}}(\mathit{cost})=?[(a\ \mathsf{U}\ a\ \mathsf{U}\ a) \wedge (b\ \mathsf{U}\ b) \wedge (\mathsf{F}\ c) \wedge (\mathsf{F}\ \mathit{Fail}\ \mathsf{X}\ \mathit{Reset})]\ ) \quad (6.5)$$

This query will ensure that optimal adversaries with respect to all possible orderings of the actions which must be performed as part of the schedule are generated. PRISM will produce one of the possible optimal adversaries (schedules). Note that the set of operators $P_1, P_2, \cdots, P_m$ may only feature two unbound operators (operators which make use of the `?` symbol), due to current implementation restrictions within PRISM. Further, in the case where multiple schedules are equally optimal only one of the possible schedules will be returned. Finally, in the case where no possible schedule exists PRISM will return an empty (zero state) DTMC.

An examination of the performance of an implementation of this technique is given in Section 9.3.3.

6.1.3 Scheduling Example

The example in Figure 6.3 illustrates a simplified scenario where a schedule must be devised for the actions of a robot arm moving materials for preparing drugs between different sub-components. This system consists of four processes, each represented as an individual *BPD* pool. The *pharmacy robot* process drives the operation of this system and makes a non-deterministic choice about which drugs to manufacture. Manufacturing each drug involves specific sequences of operations performed by separate sub components; each of these performs steps which have delays that are stochastically chosen, and various steps are annotated with rewards expressing the time and energy used.

Schedule generation requires a set of needed actions, $\mathbf{A}$. In this example, suppose the requirement was to produce two doses of A, one of B, and three of C. Note that when using the *Drug Heater* device there is also a choice between *low-power*

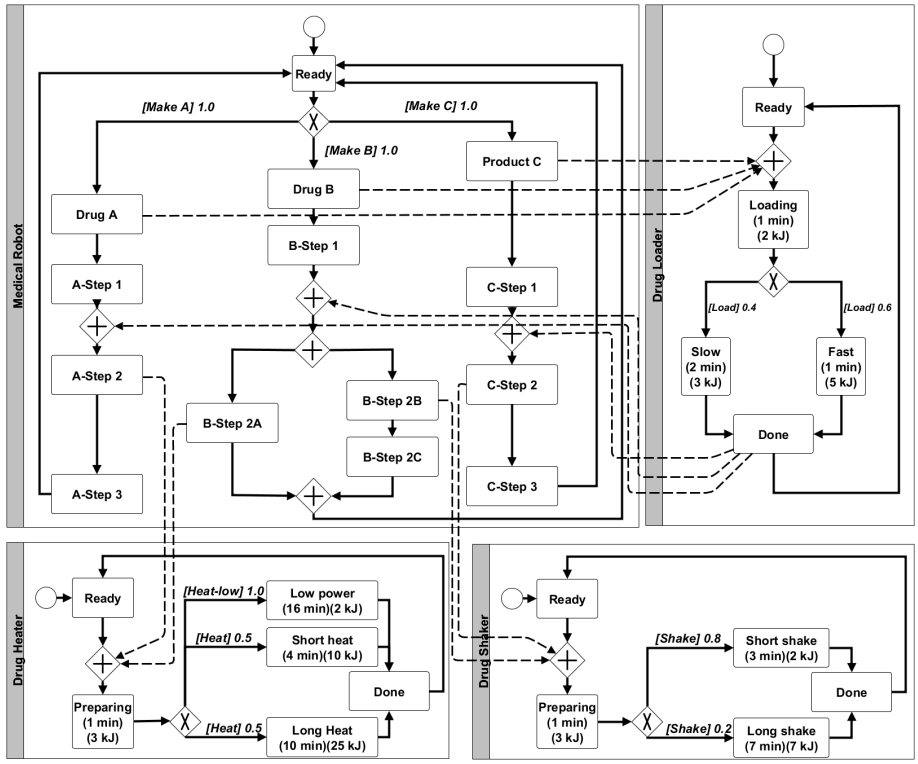


Figure 6.3: *BPMN model of a drug preparation robot.*

setting or *normal* power setting heating of the drug. These actions are not included in the set $\mathbf{A}$ of schedulable actions, so these actions will be included in the set $\mathbf{A}^{\mathtt{C}}$ and may, or may not, be included depending on what produces the optimal outcome in terms of the scheduling constraints constraints.

This relatively simple model produces a relatively large and complex statespace as shown in Figure 6.4, where the *ForceAtlas2* graph layout algorithm [111] has been employed to generate a layout which highlights the nature of the scheduling generation problem. Generating a schedule involves determining the possible paths that satisfy Q, and adversary generation in effect filters the paths of the statespace for elements which optimise the unbound queries of Q. In Figure 6.4, the initial state is represented by the black dot, and the statespace is characterised by three large loops which correspond to the manufacture of each of the three drugs. The higher complexity of the manufacture of drug B is clear in the larger number of nodes and transitions that form this loop.

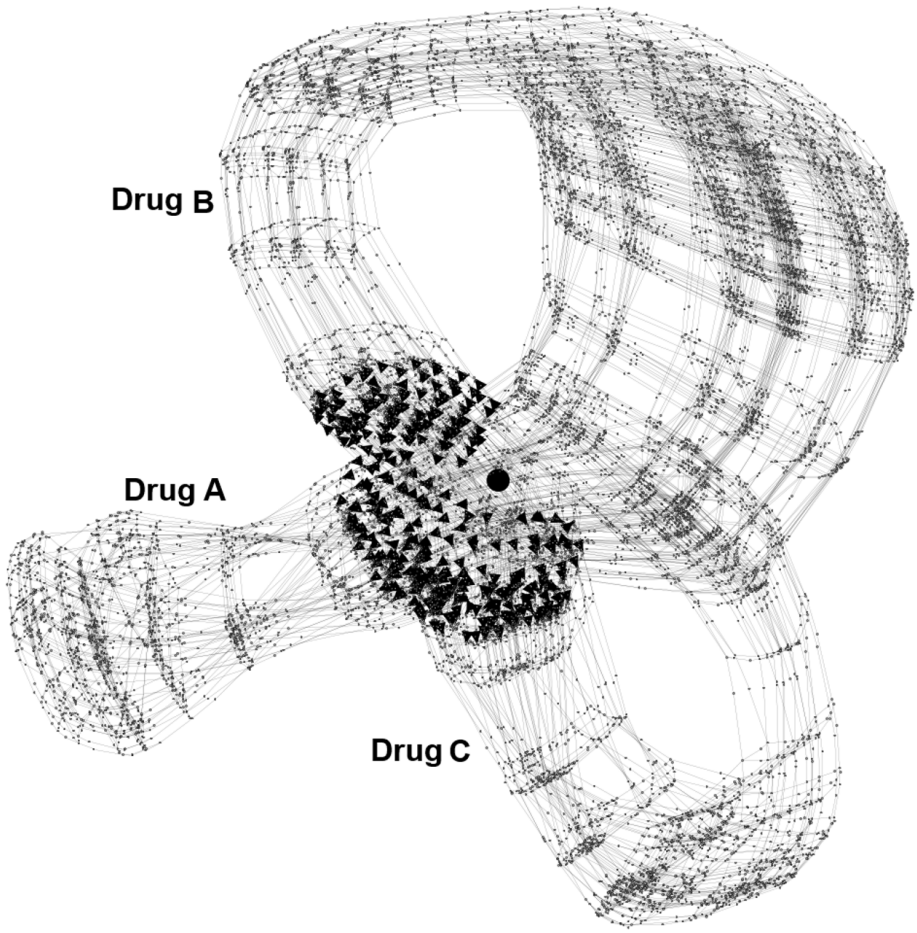


Figure 6.4: *Generated scheduling statespace for the example from Figure 6.3 (Annotations removed, 3080 States, 10999 Transitions).*

It terms of quantitative constraints it is desirable to generate a schedule under the constraints that the accumulated value of the *time* and the *energy* rewards for the chosen schedule, a path in the statespace, is the smallest possible, with equal weight being given to minimising both rewards. This involves employing the PCTL reward operator R to calculate the expected value of the time and energy reward structures for the paths that form a schedule under consideration.

An optimal schedule requires choosing the correct sequence of actions, marked by black triangles in the statespace in Figure 6.4, to reach the scheduling goal. For the specific order of drugs $\mathbf{A} = \{A \to 2, B \to 1, C \to 3\}$, this would involve interleaving, two loops through the *DrugA*, and one through the *DrugB* and three through the *DrugC* sets of path loops.

Further, in this example an additional safety requirement can be considered, namely, that there may never be any shaking taking place while a drug is being loaded as this may lead to spillage of a drug. This type of constraint can be seen as a filter on permissible paths in the statespace, excluding paths where both of these properties are simultaneously. This safety constraint set $\mathbf{C}$, is expressed in PCTL as: $\mathbf{C} = \{\mathsf{G}!([Shake] \wedge [Load])\}$

In this case applying schedule generation procedure above, there exists a unique schedule shown in Figure 6.5, with an expected mean time to completion of 37.4 minutes, using 98.3 kJ.

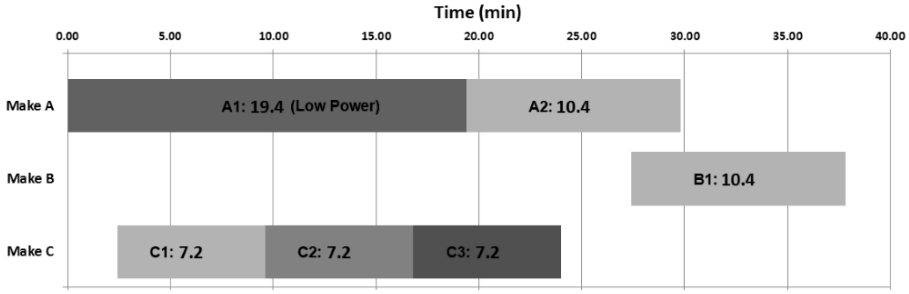


Figure 6.5: *Gantt diagram of the generated minimal time/energy usage schedule for the example from Figure 6.3 (Totals: 37.4 minutes, 98.3 kJ).*

In this solution, the robot chooses to begin production by manufacturing 1 dose of drug A and making use of the lower power heating setting in its production. Once loading is complete for drug A, manufacture of drug C is started and repeated until the loading of the 3rd dose of C. Then a second dose of A is started, and 2.4 minutes (the mean time needed to load B) before this is completed, production of B is started.

6.2 Chapter Summary

Building upon the model checking approach to the analysis of business processes, this Chapter presents a method to determine an optimal schedule (defined as a choice of actions to be performed in a business process so as to optimise properties of interest) for the execution of a business process. The schedule is created by means of the generation of adversaries which resolve points of non-determinism in a business process such that specific PCTL queries obtain their minimum or maximum value. As such the schedule generated can optimise reward values while observing constraints which encode any required execution properties.

Chapter 7

Fault Tree Analysis

"Always acknowledge a fault. This will throw those in authority off their guard and give you an opportunity to commit more."
*(*Mark Twain 1872*)*

Overview

In this chapter, Objective 3b is addressed by further extending the process graph formalism introduced in Chapter 3 to include states that capture faults which may or may not halt execution of a business processes. The injection of fault states is defined, and an algorithm presented which employs the methods of Chapter 5 for the automatic generation of fault trees from fault-annotated models. It also shows how the value of reward structures can be calculated as annotations at points of failure, defined as the combination of multiple faults, and builds a fault tree which includes the expected values of properties of interest at fault states.

The work included in this chapter was originally presented in [15] and extended in [6]. Work on combining fault analysis with optimisation techniques in Chapter 8 is presented in [2].

7.1 Errors, Faults and Failures

The Business Process Model and Notation (BPMN) standard features an error handling mechanism, where an *error* is defined as [207, page 81]:

An error is generated when there is a critical problem in the processing of an activity or when the execution of an operation failed.

When an explicit *named* error event is encountered in a standard BPMN Business Process Diagram (BPD), process flow then switches to a named *error catch event* that is then expected to handle the error. Process flow may then return to either the original process flow or to a new point in the business process. Alternatively, error handlers are attached to the boundaries of processes, and when an unexpected unnamed error occurs this generic error handler is invoked to return execution to a predictable state. This type of error handling mechanism is similar to the common approach to error handling allowed in C++, C#, Java or many other languages where code can be contained in a `try` block and where `catch` blocks then handle either specific exception types or generic errors. Resumption of normal execution can be performed by means of code in `catch` blocks or, optionally in the case of for example C# or Java, by means of a `finally` block.

While this type of error handling is well established, the use of these constructs, either in BPMN or common programming languages, requires the designer to predict the failure specific sections of a business process or code. Once this has been done, the handling of error is in essence simply a special case of a designed process flow. However, the requirement remains that errors must be predicted by a designer before error handlers can be defined to handle them.

In BPMN there has been some criticism of its poorly defined concurrency behaviour [54], when combined with the error handling mechanism in BPMN it would seem likely that multiple errors which occur in concurrent BPMN processes lead to a system state that is completely undefined. However, this case is particularly important as when multiple errors occur there is a chance that these errors can combine to cause an error that might be considerably more severe than if the each error occurred individually. For example, in a production setting, concurrent faults allowing flammable gas to escape simultaneously with a second fault producing an electrical spark may lead to a system state which is much less desirable than if each error occurs separately. Frequently when such severe faults occur the error handling mechanism for each individual may no longer be operable (e.g. turning off the gas supply may not be possible once a fire has started). Within BPMN, support for discovering such scenarios is not readily dealt with by means of the existing error handling mechanism.

This type of combined error which can occur in concurrent processes is frequently hard to predict [83], [97], [268] and is often the cause of severe, or even disastrous failures. In this chapter a technique is proposed that employs the framework for analysis of Core BPMN processes to help a designer predict possible combined errors and determine quantitative properties of the process, such as time taken or resources used, at these points of failure.

To distinguish these types of unexpected errors, which would not be caught by the standard fault hander mechanism for expected BPMN errors, the term *fault* will be used, which is defined for Core BPMN as follows:

Definition 7.1 (Core BPMN Fault)
A fault in a Core BPMN process, is an accidental condition that causes a specific task to fail to perform its required function.

The crucial situation for which the work in this chapter provides design-time support is the prediction of failures which are formally defined for Core BPMN as:

Definition 7.2 (Core BPMN Failure)
A failure in a Core BPMN BPD occurs when multiple faults combine so that an entire BPD fails to perform its required function.

The key distinction between Definition 7.1 and Definition 7.2 is the scope of the errors that occurs, *faults* are local errors in a business process while *failures* are global errors in a business process.

The work of Van der Meulen [188] highlights the broad range of competing definitions of terms such as faults, errors and failures. Core BPMN Definition 7.1 inspired by Institute of Electrical and Electronics Engineers Standard glossary of software engineering terminology [31]. Furthermore, Definition 7.2 which defines a failure for Core BPMN process is inspired by MIL-STD-721C [274], which is a US military standard which defines terms for reliability and maintainability.

7.2 Fault Tree Analysis

Originally developed in 1962 at Bell Laboratories by H.A. Watson [96], Fault Tree Analysis (FTA) has seen extensive refinement and widespread adoption and is today considered a proven and accepted reliability engineering technique. However, FTA of concurrent systems is labour-intensive, and requires a key creative step where business analysts must imagine which undesirable events can occur under which conditions [97]. FTA analysis is therefore often avoided early in a system design due to the significant effort involved in performing the analysis and ensuring its consistency with the system of interest, balanced against the likelihood that changes will be made to the design. However, correcting problems late in the development process can be costly, cause significant delays, and even require complete system redesign.

Motivated by the type of system introduced in Section 1.1, regulatory approval of these systems frequently requires the inclusion of an FTA covering the system and its immediate environment. However, construction of these trees is time-consuming and laborious with a new analysis being required whenever the system is modified or it is somehow employed in a different environment than originally intended. For this reason an automated means of generating fault trees was desired and the results in this area are presented in this chapter in line with Objective 3b.

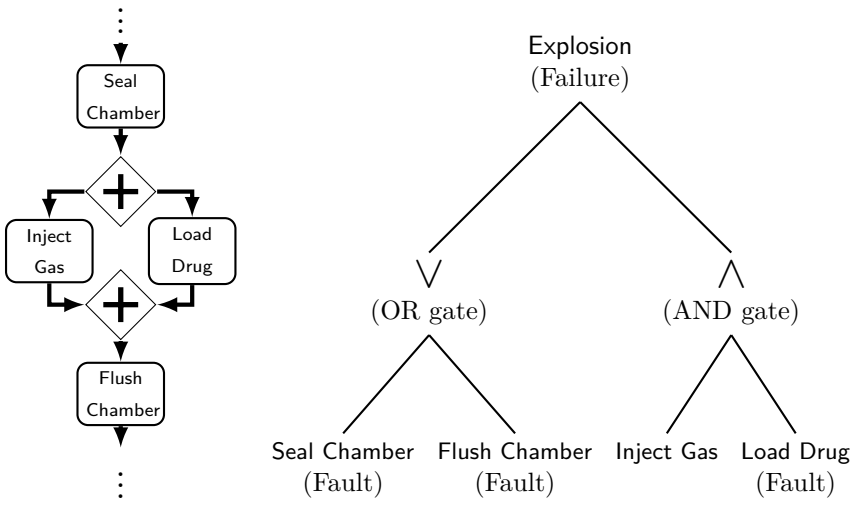
The international standard ISO13485 [143] specifies requirements for a quality management system where an organization needs to demonstrate its ability to provide medical devices and related services. Regulatory authorities worldwide are tending to adopt ISO13485 in the hope of harmonizing requirements and reducing some of the conflicting and different demands on manufacturers. Specifically, in the context of the European directives for medical devices, ISO13485 certification is required for regulatory approval, likewise Canada, Australia and a few other countries outside Europe require certification. However in the US a separate Food and Drug Administration (FDA) standard 21CFR820 [103] is required for regulatory approval. In both the case of ISO13485 and FDA-21CFR820, performing fault tree analysis is one of the key suggested techniques to gain certification. Furthermore, ISO14971 [144] sets out a set of voluntary practices pertaining to risk management for medical device companies. A key suggestion here is that employing fault tree analysis can reduce legal liability for medical device manufacturers. Due to these requirements the industrial partner in this project considered formal assistance in producing fault trees to be a key requirement in a business process analysis framework (Objective 3b).

In essence FTA is a design time analysis that seeks to help deduce possible causes of failures of a business process by suggesting possible causal chains of combinations of multiple faults that would lead to failure [268]. The results are typically represented pictorially in the form of a tree of fault modes for a given system. Figure 7.1 illustrates an FTA ((a)) for a small business process ((b)).

As is seen in Figure 7.1(b), a fault tree is a logic diagram of separate individual faults and their relationship in order to determine the probability of compound faults occurring. In this example it is imagined that a failure to load the drug correctly may lead to a spark and as the gas is flammable this leads to an explosion, likewise if the chamber is not sealed or fails to be flushed after use, the gas escapes the chamber and an explosion is also likely.

In Figure 7.1(b) the root node of a fault tree is a possible failure and each path from a terminal node to the root describes a specific combination of faults that could lead to failure. Within a fault tree every other level of the tree contains combination operators which are expressed using logical operators (AND, OR, etc.) which indicate how the named faults in the level below must combine to contribute towards a failure.

FTA is fundamentally used to establish the pathway(s) to the root cause of a failure and hence investigate or refine elements of a system so as to try and mitigate or prevent failure. Repeated fault tree analysis is intended to help minimise the causes of failure and check if intended improvements will fully resolve the issue and not lead to other issues (i.e. solve one problem yet cause a different problem). Fault Tree Analysis is an effective tool for evaluating



(a) Sterile Drug Loader (b) Sterile Drug Loader Fault Tree

Figure 7.1: *Illustration of fault tree for a sterilising drug loader (a), where a flammable gas is used to ensure sterility within a sealed chamber. The fault tree analysis of this model (b), suggests a number of causal chains which can lead to a failure (explosion).*

how multiple factors affect a given issue and is considered useful both for risk assessment, in developing monitoring programs and for directing component testing [97], [268].

While typical manual FTA relies on an experts' understanding of a process to identify causal factors, the intention of *automated* fault tree analysis is to allow system designers to focus only on modelling a system (as shown in Definition 7.1) and imagining possible local faults that may occur during execution of the tasks performed as part of a business process. The methods presented in this chapter allow system designers to make such annotations directly to Core BPMN models and then automatically perform FTA. The probability of failures (as defined in Definition 7.2) occurring can then be automatically calculated based on both the probability of individual faults occurring and inherently stochastic behaviour that a business may exhibit in the course of its correct execution.

Automated FTA seeks to build maximal fault trees which include a wide range of potential faults which may, or may not, be possible in a real system. Furthermore, determining the value of reward structures at points of *failure* can guide testing of such systems. It provides upper and lower bounds on *time to failure* or the

amount of some resource that will be consumed by the time a failure occurs. This can be useful in cases when some faults cannot be eliminated. A focus can instead be made on ensuring failure happens as late, or early, as possible, or alternatively ensure that resources consumed at the points of failure are minimised.

7.3 Fault Tree Analysis via Model Checking

Ideally, safety engineering starts during the early design of a system. Even at an early stage in the design process, business processes may exhibit a high degree of complexity [251], and formal modelling of the system typically captures the ideal, fault-free, conception of the system. At this early stage, an FTA constructed for a business process is based on an informal description of the underlying system [97], or requires modelling the system in a separate FTA specific modelling language [41], [174]. This makes it difficult to check the consistency of the analysis, because it is possible that causes are noted in the tree which do not lead to the failure (incorrectness), or that some causes of failure are overlooked (incompleteness).

Liggesmeyer and Rothfelder [174] coined the term *formal risk analysis*, when they developed an approach for automatically generating a fault tree from finite state machine-based descriptions of a system where the generated fault tree is complete with respect to all failures assumed possible. In this chapter a similar approach is presented that exploits the analysis defined in Chapter 5 to generate fault trees which reflect the base stochastic behaviour of the system from the generated statespace of a Core BPMN model as shown in Figure 7.2.

The design of the analysis is shown in Figure 7.2, the dark grey boxes are the additional material that must be supplied by a user in addition to a Core BPMN system model with annotations (light grey boxes). Fault state injection is handled through functions which mark states in a Core BPMN model as having the potential to exhibit a fault (Section 7.4). Note that these additions are simply annotations to an existing model and require no structural changes to the model. Reward structures are not strictly needed to simply perform a qualitative FTA. However, if data is associated with a system model this can readily be mapped to the resulting fault tree, which allows the expected values of properties of interest, such as time, power usage or financial cost, to be included in the fault tree.

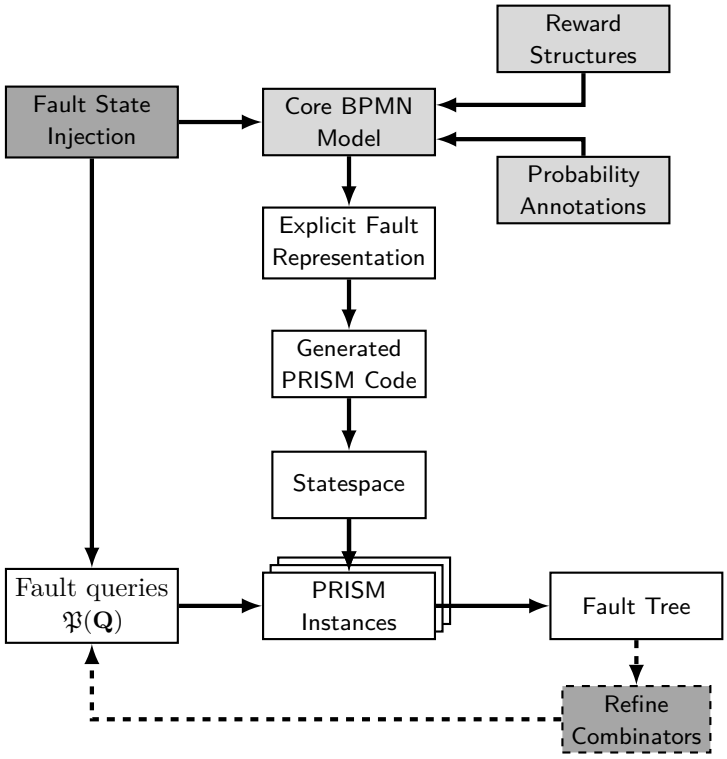


Figure 7.2: *Overview of the Core BPMN analysis framework approach to fault tree generation (grey boxes mark inputs needed, with dark grey being the additional fault state injection).*

A fault annotated model is then converted into a representation where fault states are made explicit. Faults are defined in terms of standard Core BPMN probabilistic gateways as defined in Definition 3.14 and standard Core BPMN task elements. Hence, conversion to an explicit fault representation simply involves replacing fault annotated states with fixed blocks of Core BPMN elements.

A key strength of this approach is that a statespace for the model, extended with faults, need only be generated once. Following this, the computationally most expensive step, any change in how faults are combined (dashed lines), only requires performing new Probabilistic Computation Tree Logic (PCTL) queries over the existing statespace. Only a change to which states are marked as having the potential for fault requires new statespace generation. The ability to perform multiple queries over a single statespace is the key to making this approach

computationally feasible, by parallelising the checking of separate fault queries, the main computational expense is statespace generation which must only be performed once.

7.3.1 Related Work

The approach developed in this chapter enhances the original formal risk analysis concept of [174] with direct translation from a business process modelling language (Core BPMN) by allowing for the addition of data values at points of failure in a fault tree. For the cost of a small amount of additional data added to a workflow model, it becomes possible to automatically derive rich fault trees with no further user interaction needed. Compared to the symbolic model checking employed by Liggesmeyer and Rothfelder [174] the use of quantitative probabilistic model checking allows for fault trees to be constructed where both quantitative properties of the system at points of failure can be determined and the inherent stochastic nature of the underlying system is accounted for when determining the probabilities of failure.

Work by Banach and Bozzano [41] allows for the automated generation of fault trees. However, their work focuses on digital circuits, which must be modelled in a custom modelling language for automated generation to be possible. Further, their work deals with digital circuits of an entirely deterministic nature which does not feature stochastic behaviour. In order not to add a further source of error, the models constructed by the framework presented in this thesis are automatically translated into fault trees with no remodelling required.

Work by Thums and Schellhorn [271], and by Xiaocheng et. al. [109], employs model checking to verify the correctness of an FTA and to perform model checking of fault trees. In both cases model checking is used to determine the feasibility of complex faults, described using Computation Tree Logic (CTL), within large fault trees. However, in both cases, the source model for the FTA requires custom modelling of the system of interest and does not incorporate the use of rewards in the analysis.

An interesting approach, where fault trees are generated from Duration Calculus (DC) [296] models via model checking is suggested by Schäfer [257] where fault trees are generated from DC models. In his work a very restricted subset of DC, known as DC implementables [210], is employed. This variant severely restricts the use of negation, and therefore limits the type of system it can model. However, the approach does provide a fully automated way of going from DC models of a system to fault trees and a number of timing properties at points of failure are accounted for.

These last two approaches to fault tree construction are the most similar to the approach developed in this thesis and as such the approach presented does not fundamentally differ from the original approach of Liggesmeyer and Rothfelder [174] but does provide significant enhancements by providing an automated mapping from stochastic business processes to fault trees and including quantitative properties in the resultant fault tree.

7.4 Modelling Faults

To allow for the automated generation of fault trees, process graphs are extended to include two types of faults. These are known as *fault-stop* and *fault-continue* faults. In terms of a Core BPMN model these take the form of annotations which denote which type of fault can occur and its associated probability. The semantic interpretation of these fault states is described in terms of Core BPMN elements and described in the following subsections.

It may also be desirable to have a model of faulty decision gateways in a workflow and this can be modelled simply by redefining gateway flow probability function $\mathcal{P}$ associated with a gateway, to include a number of branching probabilities.

It should be noted that the types of faults modelled are intended to be introduced at the design stage, before the development of mechanisms in a business process that allow for corrective action or detection of faults has been initiated. Instead, in line with the FTA paradigm, the intention is to explore the potential effects of faults and how they might propagate and thereby help determine where detection and corrective actions can be of most benefit.

7.4.1 Fault-Stop Faults

To ease the task of building fault-tolerant systems, most designers try to ensure that a malfunctioning system halts before it can cause further damage. This property is known as halt-on-failure [261] (not to be confused with the definition of failure given in Definition 7.2) and is the main motivation for the *fault-stop* model of faults. In next section Section 7.4.2 a broader category of faults are introduced which model cases where a fault does not lead to an execution stop, as such the *fault-stop* faults introduced here can be seen as a special case of the faults in the next section.

Formally the addition of fault-stop faults to Core BPMN processes is given in Definition 7.3:

Definition 7.3 (Fault-Stop Task Fault Injection Function)
For a task $t_n \in \mathbf{T}$ in a BPD, the partial function

$$\vec{\mathcal{E}} : \mathbf{T} \times [0,1] \to (\mathbf{T} \times \mathbf{T}) \times [0,1]$$

adds a fault-stop execution sequence to a BPD as follows:

$$\vec{\mathcal{E}}(t_n, p) = \begin{cases} t_{n-1}\mathcal{S}\vec{t_n} & \text{with probability } p \\ t_{n-1}\mathcal{S}t_n & \text{with probability } 1-p \end{cases} \tag{7.1}$$

Application of Definition 7.3 to add *fault-stop* behaviour to a task is illustrated in Figure 7.3. Here, after task t_{n-1} has been performed, then with a probability of $P_{\vec{t_n}}$ the task t_n is not performed, and instead a transition $t_{n-1}\mathcal{S}\vec{t_n}$, to a state $\vec{t_n}$ representing the process halting (deadlocking) during execution of t_n is made. The set of all fault-stop tasks is denoted as $\vec{\mathbf{T}}$.

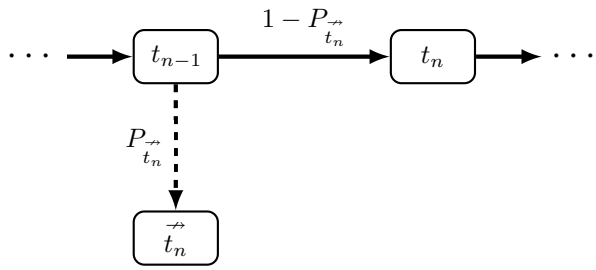


Figure 7.3: *Illustration of Fault-Stop behaviour in Core BPMN processes.*

Fault-stop behaviour is frequently desirable in production systems as the halting (deadlocking) behaviour should ensure that further damage does not occur and a plan can be devised for recovery. However, for fault-stop behaviour to be feasible it requires that faults can be detected, which may not be the case. Additionally, in the case of concurrent systems, it may not be desirable to stop a process when other processes depend on tasks which can still be executed correctly by the faulty process.

Employing the denotational semantics model of business process execution from Section 3.5 the trace produced by injection of a *fault-stop* fault is to truncate the original non-faulty execution trace. Given a process modelling a sequence

of tasks $t_1 \rightarrow t_2 \rightarrow t_3$ with an execution trace $< t_1, t_2, t_3 >$ the application of $\overset{\frown}{\mathcal{E}}(t_2, p)$ produces the trace $< t_1, \overset{\frown}{(}t_2) >$ with probability p and the trace $< t_1, t_2, t_3 >$ with probability $1 - p$. Note that the execution halts only for the execution of a single process (pool) and the trace sequence may subsequently feature further events from other processes.

7.4.2 Fault-Continue Faults

Undetectable faults will be termed *fault-continue* faults. The behaviour of these faults is such that a permanent fault has occurred, which will not be corrected or even detected in the later execution of the business process. These faults exhibit more complex behaviour in that a business process does not halt, and execution of the business process proceeds, despite a fault occurring in a specific task. Formally the addition of fault-continue faults to a Core BPMN model will be described by means of Definition 7.4:

Definition 7.4 (Fault-Continue Task Fault Injection Function)
For a task $t_n \in \mathbf{T}$ in a BPD, the partial function

$$\overset{\frown}{\mathcal{E}} : \mathbf{T} \times [0,1] \rightarrow (\mathbf{T} \times \mathbf{T}) \times (\mathbf{T} \times \mathbf{T}) \times [0,1]$$

adds a fault-continue execution sequence to a BPD as follows:

$$\overset{\frown}{\mathcal{E}}(t_n, p) = \begin{cases} t_{n-1}\mathcal{S}\overset{\frown}{t_n}\mathcal{S}t_{n+1} & \textit{with probability } p \\ t_{n-1}\mathcal{S}t_n\mathcal{S}t_{n+1} & \textit{with probability } 1-p \end{cases} \tag{7.2}$$

Application of Definition 7.4 to add fault-continue behaviour to a task is shown in Figure 7.4. In this case, after task t_{n-1} has been performed, then with a probability of $P_{\overset{\frown}{t_n}}$ the task t_n is not performed and instead a sequence of transitions $t_{n-1}\mathcal{S}\overset{\frown}{t_n}\mathcal{S}t_{n+1}$ passing through state $\overset{\frown}{t_n}$, representing the task being performed in some faulty, but not deadlocking, fashion is performed. The set of all fault-continue tasks is denoted as $\overset{\frown}{\mathbf{T}}$.

Employing the denotational semantics model of business process execution from Section 3.5 the trace produced by injection of a *fault-continue* fault is a simple change of the original non-faulty execution trace. Given a process modelling the execution of sequence of tasks $t_1 \rightarrow t_2 \rightarrow t_3$ with an execution trace $< t_1, t_2, t_3 >$ the application of $\overset{\frown}{\mathcal{E}}(t_2, p)$ produces the trace $< t_1, \overset{\frown}{t_2}, t_3 >$ with probability p

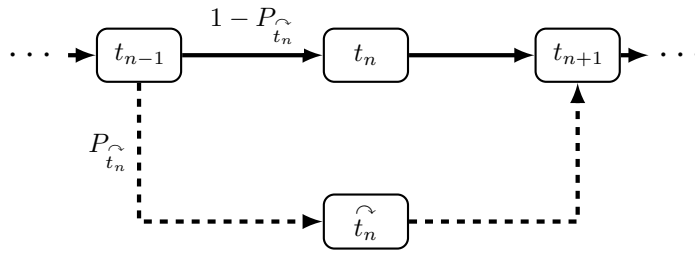


Figure 7.4: *Illustration of Fault-Continue behaviour in Core BPMN processes.*

and the trace $< t_1, t_2, t_3 >$ with probability $1 - p$. Thus, the introduction of *fault-continue* faults produces additional traces for the execution of business process but preserves the ordering of non-faulty events when executing a business process. The small change in the trace of execution allows the same methods developed in Section 5.4 to be employed to determine the total possible behaviour of model in the same fashion as if the model did not contain faults.

7.5 Fault Tree Analysis in a Stochastic Setting

Traditionally fault trees are defined as illustrated in Figure 7.1(b) where each path through a fault tree defines a combination of AND/OR dependencies which may lead to additional faults as defined in Definition 7.1 or following Definition 7.2 complete system failures. In this setting the specific FTA probabilities for non-terminal nodes can be derived from each terminal node by means of common probability calculations (e.g. given faults A and B, $P(A \vee B) = P(A) + P(B) - P(A \wedge B)$).

In the case of core BPMN, the probability of fault states occurring in a stochastic Core BPMN BPD depends on the inherent stochastic structure of the existing BPD where the occurrence of faults may require iterations of loops or the specific interleaving experienced in concurrent parts of a BPD to arrive at combined failure states. This means that a suitable FTA for a stochastic Core BPMN model may be considerable more complex than traditional FTAs described in Section 7.2. In this setting quantitative stochastic model checking provides an ideal tool to determine the probabilities of arriving in combined fault states within Core BPMN models. The resultant fault trees provide a convenient symbolic representation of the combination of faults causing a failure as defined in Definition 7.2, and they are represented as a parallel or sequential combination of logical *AND* and *OR* gates.

The total application of $\vec{\mathcal{E}}$ and $\overset{\frown}{\mathcal{E}}$ produces the maximal set of possible fault states, where all tasks can exhibit both *fault-stop* and *fault-continue* behaviour. Such a total fault tree can be a useful starting point to help suggest all possible routes to failure. Once this has been done it is beneficial to restrict the set of tasks believed to be prone to failure to eliminate *false-positive* faults that the designer knows are impossible in the business process. This can be achieved by choosing to only apply $\vec{\mathcal{E}}$ or $\overset{\frown}{\mathcal{E}}$ to specific tasks in a business process.

A maximal fault tree takes the general form shown in Figure 7.5. In such a fault tree every possible combination of pre-conditions for every combined fault state is represented. Note that this fault tree contains a large number of redundant nodes, as elementary fault occurs n times as a leaf node, where n is the total number of elementary faults. However, in practical use significant sections can readily be removed during manual analysis as they may represent faults which are improbable within the specific business process under consideration.

Note the following fairly common definition will be used to define the subset, elements of the powerset of fixed size k of a set X as $\mathfrak{P}_k(X) = \{A \in \mathfrak{P}(X) : |A| = k\}$.

In Figure 7.5 terminal nodes consist of individual states, which may be represented more than once, which are then combined to construct every possible combination of sub-sequences of individual faults. The top node is the ultimate failure state, which is a combination of all possible faulty tasks, i.e. the largest element of the powerset $\mathfrak{P}(\mathbf{T})$ where $\mathbf{T} = \overset{\frown}{\mathbf{T}} \cup \vec{\mathbf{T}}$. The nodes in between capture every possible *OR-gated* combination of sub-states that can lead to this failure condition. Each level in a fault tree consists of all possible partitions of elements N of a node above the size of $|N| - 1$ combined with the singleton element not included in each partition.

It should be noted that within the fault tree various sub-states will be repeated along different paths, but every sub-state will be an element of $\mathfrak{P}(\mathbf{T})$. This, as can be seen in Figure 7.5, allows substantial parts of the fault tree to be collapsed and replaced by a pointer to an already computed sub-tree. This approach avoids both unnecessary work in terms of the generation of annotations and simplifying the structure to make it more readily comprehensible. In a practical implementation, as presented in Chapter 9, an approach where the elements presented can be expanded or collapsed makes the representation of the tree considerably more compact.

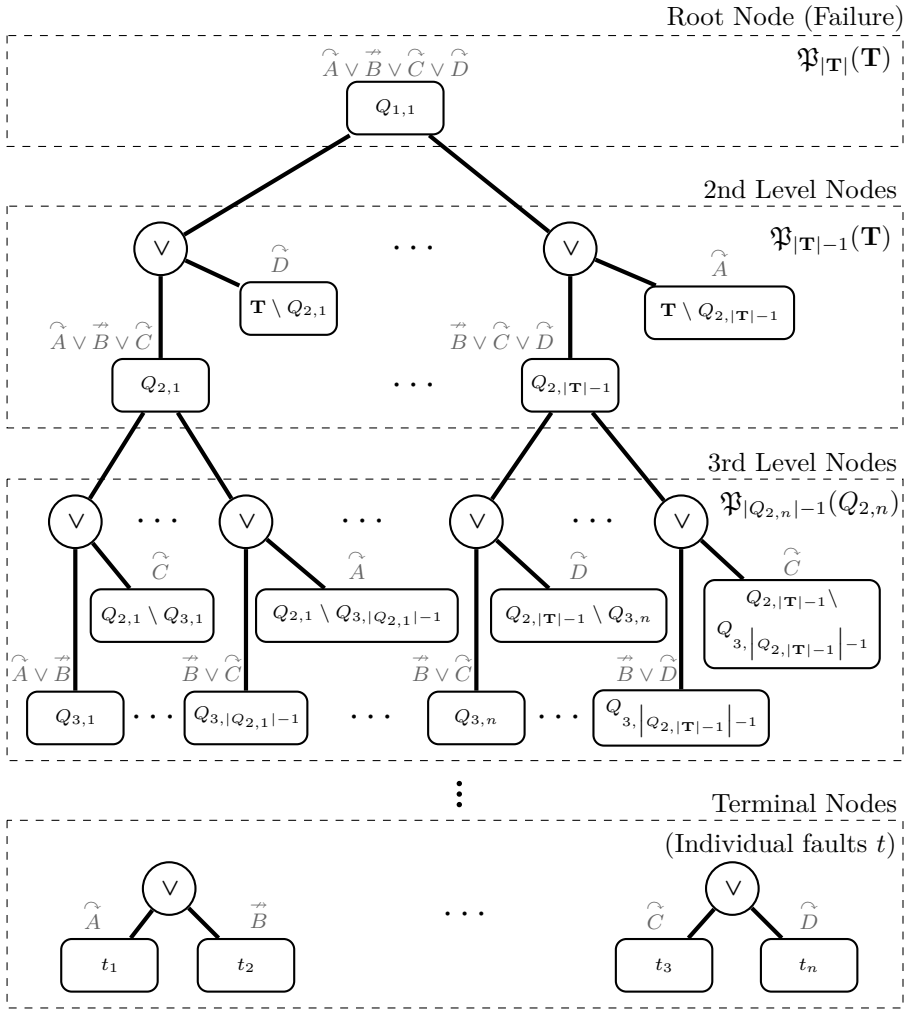


Figure 7.5: *Illustration FTA generation by means of model checking using the method in Algorithm 4 below. Grey text shows example fault states for each node.*

Producing a fault tree involves systematically building all possible chains of one of more fault-continue faults that can occur in the business process, possibly combined with a final *fault-stop* fault. It is therefore necessary to examine all possible business processes' executions.

7.6 Generating Fault Trees for Core BPMN

Figure 7.5 defines the structure of a fault tree for a Stochastic Core BPMN model. Such a tree can be dynamically constructed by means of an algorithm such as seen in [80]. However, the complexity of tree generation is fundamental to any fault tree construction and involves the computation of a powerset with a well established complexity bound of $O(2^n)$, where n is the number of elements in the set. Therefore in the following analysis of Algorithm 4 focus will be on the complexity of the additional work needed to calculate node probabilities and reward values.

Producing a fault tree involves systematically building all possible chains of one of more fault-continue faults that can occur in the business process, possibly combined with a final *fault-stop* fault. For a stochastic Core BPMN BPD, annotated with rewards $\mathcal{R}$ and error annotations $\overset{\frown}{\mathbf{T}}$ and $\overset{\rightharpoonup}{\mathbf{T}}$, Algorithm 4 employs the developed framework for the analysis of business processes to determine quantitative values for nodes in the tree by means of executing PCTL queries. Specifically, the values for probability and reward values for each node in a fault tree is determined by Algorithm 4 in the following fashion:

In Algorithm 4 a fault tree is constructed by performing queries of the probabilities and values of *rewards* of interest, in each possible fault, and combined fault state, in $\mathfrak{P}(\mathbf{T})$. Each of these calculations, both for probabilities and any relevant reward structure, is then matched to the appropriate state or states, represented by nodes within the fault tree. In lines 14 and 15 the index of node is given as $|q|$ which denotes a unique label generated from the states of a given query $q \in \mathbf{Q}$.

It should be noted that some combinations of faults within the maximal fault tree will have a probability of 0 and it is convenient to omit such states from the fault tree. When used, the bound c places a limit on the number of transitions of the model to check; this is useful in cases where a model has the potential to have an infinite loop and upper bounds of probabilities (using max query modifier) produce infinities. As no business process can be expected to run forever this allows a user to place limits on the expected number of execution steps that will be performed in a business process.

Algorithm 4 is guaranteed to terminate as it operates over a finite set. It has a complexity bound of $O(2^n)$, where n is the size of the set $\mathbf{T}$ which, while large, is still feasible as all queries are performed on the same state space generated by the Probabilistic Symbolic Model Checker (PRISM) model checker, and the construction of the statespace by PRISM still dominates the computational

Algorithm 4: BPMN Fault Tree Generation algorithm.

Input: Sets $\overset{\frown}{\mathbf{T}}$ and $\overset{\rightarrow}{\mathbf{T}}$, and a fault tree structure FT of nodes n_x.
(An optional bound c on the depth of checking)
Output: A probability and reward annotated OR-Gated fault tree FT.

```
1  T ← 𝔓(T⌢ ∪ T→)                          // Combine all fault states
2  Q ← ∅                                    // Intialisation of Q
3  forall the t ∈ T do //                   For each element t in T
4  |  forall the s ∈ t do //  Get the individual states s of each t
5  |  |  Q ← Q ∨ s                          // Combine states s using OR-gates
6  forall the q ∈ Q do //                   Build PRISM queries based on q
7  |  p ← P_max=?[F_<=c q]                  // Determine the state probability
8  |  if p > 0 then //                      If q is possible
9  |  |  R ← ∅                             // Intialisation of R
10 |  |  forall the l ∈ Labels do
11 |  |  |  R ← R ∪ R^l_max=?[F q]          // Determine reward upper bounds
12 |  |  |  R ← R ∪ R^l_min=?[F q]          // Determine reward lower bounds
13 |  |  forall the n ∈ FT do
14 |  |  |  n_|q| is associated with p      // Assign probability to node
15 |  |  |  n_|q| is associated with R      // Assign rewards to node
16 return FT
```

burden of adding probabilities and reward values. Furthermore, note that this complexity bound need not be in addition to the fault tree generation itself as calculation of rewards and probabilities can be combined with tree construction. Fundamentally however fault tree construction has large complexity and is best suited to analysis of systems where only a small number of tasks are considered as being able to exhibit a fault. Note that a fault tree is intended for use by a business practitioner who must consider the real-world feasibility of proposed faults and therefore is not commonly applied to systems which can exhibit a large number of distinct faults.

The choice of gates to combine nodes is by default chosen to be *OR* gates, so as to build a fault tree with the greatest likelihood of producing a failure. However, the choice to combine separate fault states by means of *OR*-gates is only the default combinator. In a practical implementation of these techniques, as presented in Chapter 9, once an initial fault tree has been constructed it is possible to select and modify gates to exhibit *AND*, *OR* or *XOR* behaviour. Once these

changes have been made, Algorithm 4 is run again to compute new probability and reward values, with the key change in line 5 being to choose the appropriate $\vee$ or $\wedge$ operator.

The choice of *OR* as the default combinator is motivated by seeking to provide a maximal fault tree where the minimal possible set of causes can lead to a failure (i.e. only one fault at the base of a fault tree will propagate to become a failure). It is the work of a business process designer to restrict the FTA produced by this technique to more closely resemble the system of interest. However, the default fault tree generated will be a superset of all the more restricted failure modes.

A practical implementation of this analysis is presented in Section 9.3.4.

7.7 Example

An example of a BPMN workflow that is simple enough, so that the generated fault tree is not too large for presentation, is shown in figure 7.6.

In this example there are two BPMN pools, composed as a BPD modelling a security camera and control centre which responds to images sent from the camera. The camera process begins by entering a ready state and then proceeds to perform a capture image task, which is annotated with reward structures to indicate the time taken (60 seconds) and memory consumed (1 unit of memory).

After that, the camera analyses the image to determine if there is suspicious activity. This process is similarly annotated with a time reward, but also marked with a 0.3 probability of experiencing a fault-continue behaviour (indicated by the + symbol), e.g. the image buffer could be corrupted. The camera process proceeds to make a choice about whether the image is suspicious or not. If the image is not suspicious, then, with a probability of 1, the camera process loops back to the ready state. If the image is suspicious, a message is sent to the control system, a task which may also be faulty.

The control system, after entering its ready state, will only progress to the receive image state when an image has been received. The image is then reviewed by a human operator who decides whether there is an actual problem. In the case where there is a problem, a non-deterministic choice is made between calling a guard or calling the police, with probabilities of 0.7 and 0.3, respectively. In the case where a guard is sent there is a possibility of fault-stop behaviour, indicated by the ! symbol, and an associated probability of 0.2 that the guard does not respond to the call.

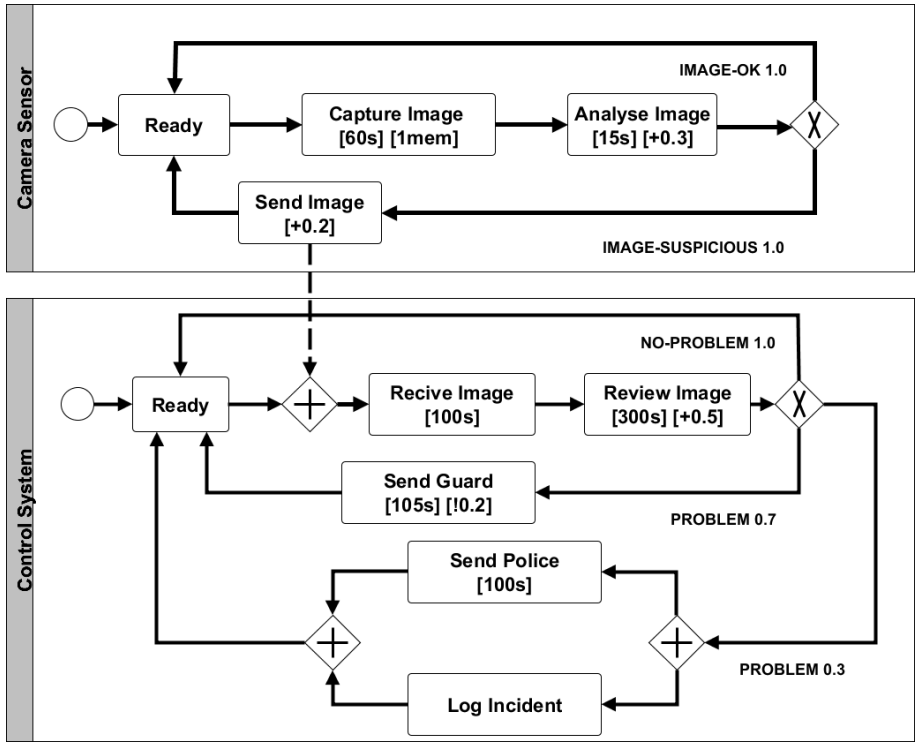


Figure 7.6: *BPMN model of a smart security system, which contains faults.*

The fault annotations added to the model in Figure 7.6 can be seen, by definitions Definitions 7.3 and 7.4, to add extra states and gateways to a Core BPMN model. An explicit representation of these failure states in shown in Figure 7.7. This illustrates the fashion in which fault annotations are added to the model in terms of standard Core BPMN elements.

For this system, determining the fault tree of the possible combined faults that could occur results in obtaining knowledge about how this system can fail to maintain security. The fault tree shown in Figure 7.8 determines maximum probabilities, the minimum time taken, and the expected amount of memory used for every possible fault, and *OR* combined failure, that can potentially arise.

The combination of failure probabilities combined with quantitative system performance data determined by this analysis allows business process designers to determine the impact specific tasks in a business process may have on the overall system reliability. Faults in tasks which make a substantial contribution

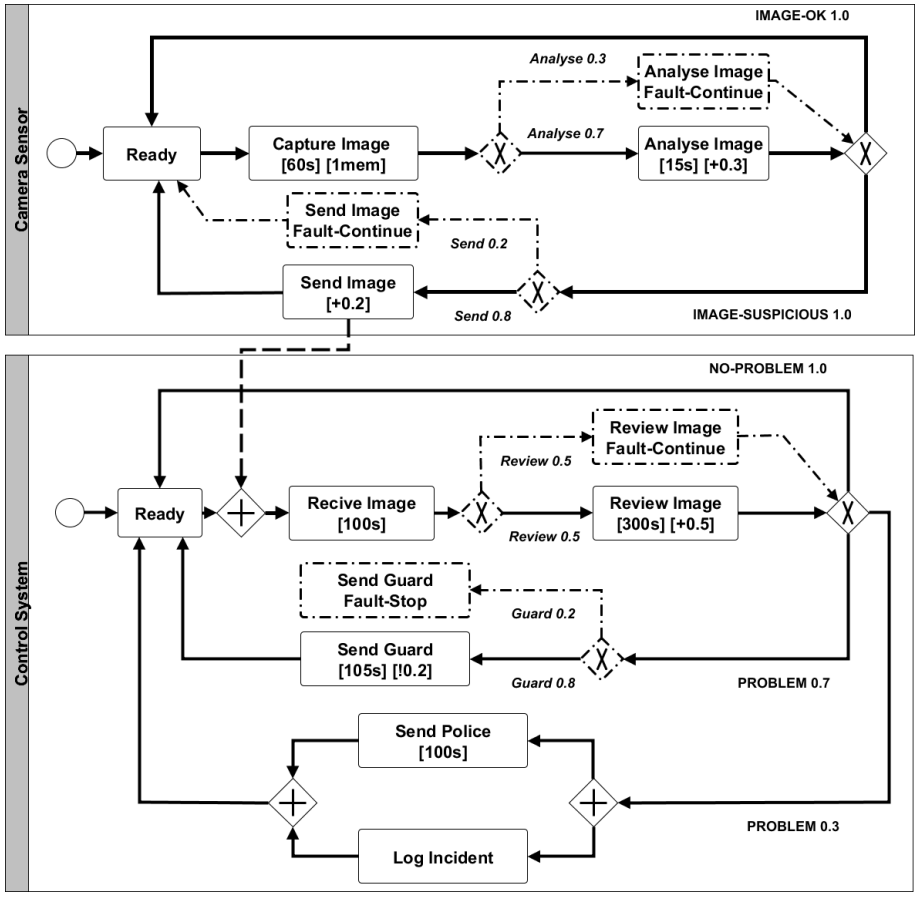


Figure 7.7: *BPMN model of a smart security system, where fault states are explicitly added (dot-dashed elements).*

to the probability of failure can be identified and then altered to increase the reliability of the business process. This can be achieved either by eliminating the task, replacing the task with one that has a lower probability of encoring a fault or restructuring the BPD so as to reduce a potential fault task's impact when it fails. Further, a fault tree can be crucial in deciding where to introduce safeguards within a business process, while these safeguards may also have a chance of incurring a fault, generating a new fault tree will help determine to what extent they reduce the chance of failure.

For the model shown in Figure 7.6, determining the fault tree of the possible combined faults that could occur results in obtaining knowledge about how this system can fail to maintain security. The fault tree shown in Figure 7.8 determines probabilities, the minimum time taken, and the expected amount of memory used for every possible failure, and combination of failures, encoded in the system model.

Note that the structure of the fault tree in Figure 7.8 is combined by *OR*-gates, hence the probability of failures (i.e. notes where more than one fault is present) increase along paths from the terminal nodes to the root node. However, the probabilities do not increase in the set union fashion that is traditional for an *OR*-gate (i.e. $P(AorB) = P(A \cup B) = P(A) + P(B) - P(A \cap B)$). Instead probabilities depend on the underlying stochastic structure of the BPD. For example, if one considers the failure state $\overset{\curvearrowright}{AI}, \overset{\curvearrowright}{SI}, \overset{\nrightarrow}{SG}$, the probability of failure is the same as for the root node failure $\overset{\curvearrowright}{AI}, \overset{\curvearrowright}{SI}, \overset{\nrightarrow}{SG}, \overset{\curvearrowright}{RI}$. This is due to the fact that the missing fault $\overset{\curvearrowright}{RI}$ can earliest occur after the other three faults have occurred. Hence, a designer has certain knowledge about the likely way a series of faults will unfold to cause a failure. This information can be crucial when choosing where in a system to implement countermeasures.

The quantitative data included in the fault tree in Figure 7.8 can help guide testing as it suggests bounds on the expected time taken to reach a failure state. This information can be used to experimentally verify if a failure proposed by the FTA is actually realised in an implementation of the system.

7.8 Chapter Summary

In this chapter the concept of fault tree analysis is introduced and a mechanism is defined for marking states as exhibiting *fault-stop* and *fault-continue* behaviour. This allows for employing model checking for automatic generation of fault trees by building PCTL queries determining the reachability of all fault and combination of fault states. Simultaneously determining the values of reward structures at the points of faults is readily possible. This automated approach to fault tree analysis accounts for the underlying stochastic behaviour of a business process and can be done directly from a model of a business process with the only additional information needed being the definition of states that are likely to exhibit a fault.

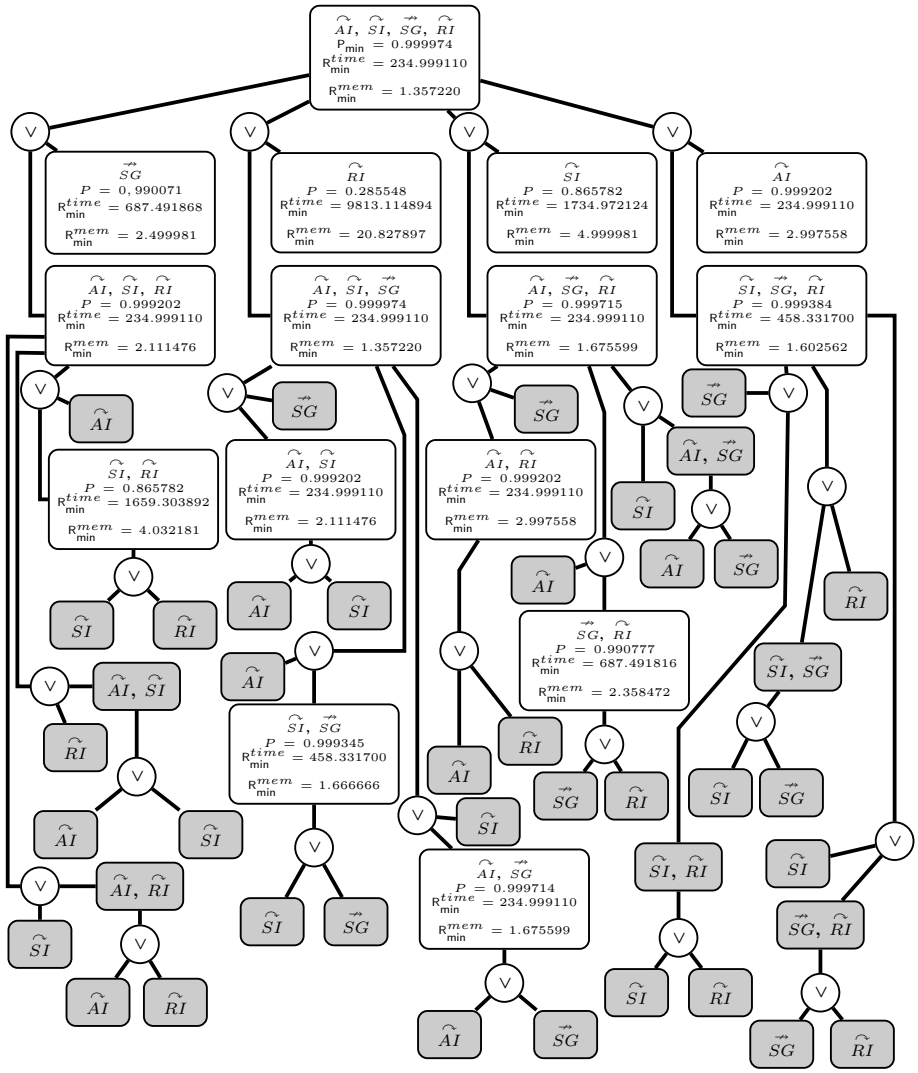


Figure 7.8: *Generated OR-gated fault tree for the example shown in Figure 7.6.* $\overset{\frown}{AI}$ *denotes a fault-continue fault in the* analyse image *task,* $\overset{\frown}{SI}$ *a fault-continue fault in sending the image,* $\overset{\rightarrow}{SG}$ *a fault-stop when sending a guard and* $\overset{\frown}{RI}$ *denotes a fault-continue fault in reviewing an image. (Grey states denote duplicates of existing states)*

Chapter 8

Optimisation of Business Processes

"In the struggle for survival, the fittest win out at the expense of their rivals because they succeed in adapting themselves best to their environment." (Charles Darwin 1859)

Overview

This chapter addresses Objective 3c. A method for the automated optimisation of business processes, which permits the effective integration of new technology or the improvement of existing processes, is presented. Goals for optimisation with respect to both functional and non-functional properties required of an optimised business process can be specified. Using these goals, an algorithm is presented which takes these goals and existing business processes as inputs to produce an improved business process.

The algorithm is an iterative evolutionary algorithm which generates variants of an existing business process, and, at each iteration, determines if they meet the functional requirements. If a higher fitness score is achieved with respect to the weighted non-functional properties, then this variant is used as the basis for deriving the next generation of variants in the next iteration. The algorithm's core parameters and how they influence the process are discussed and a small illustrative example is given.

The work in this chapter was originally presented in [13] and refined in [7].

8.1 Business Process Optimisation

Frank Gilbreth's seminal 1921 paper [113] *Process Charts - first steps to finding the one best way to do work* is credited with developing the first method for documenting process flow. The later development of the ideas originating in his paper have been crucial in informing today's concept of *Business Process Modelling*. However, developing business processes is today still predominantly a manual activity. Business processes are analysed by hand and improved configurations are found by a process of trial and error, often taking many years to arrive at an optimal practice. Key factors which prevent the realisation of optimal efficiency gains are: adapting to disruptive technology [112], resistance to change[161], and a lack of understanding that, while possible, simply replacing one function in an established workflow with a new technology does not lead to maximal gains.

Pioneered by Hammer in 1990 [126] the dominant approach today to business process *improvement* is that of Business Process Re-Engineering (BPR). This approach focuses on the analysis and adjustment of workflows and business processes within an organization. The BPR approach is aimed at helping organizations fundamentally rethink how they do their work in order to dramatically

improve competitive advantage. A survey conducted a few years after the BPR concept was introduced showed that as many as 60% of the Fortune 500 companies claimed to have initiated re-engineering efforts [127].

BPR emphasizes a holistic focus on business objectives and how processes are related to them, encouraging full-scale recreation of processes rather than iterative optimization of sub-processes [126]. Hammer's central claim was that most of the work being done does not add any value for customers, and this work should be removed, not simply accelerated through automation. This statement implicitly accused managers of having focused on the wrong issues, namely that technology in general, and more specifically information technology, has been used primarily for automating existing processes rather than using it as an enabler for making non-value adding work obsolete.

The central idea of the BPR framework was therefore to perform an assessment of mission and goals of an organisation and re-engineer an organization's business processes by decomposing them into specific activities which could be modelled, measured and ultimately improved. The intention is that many business processes can be completely redesigned or eliminated altogether, as an organization may find that it is operating on questionable assumptions, particularly in terms of the wants and needs of its customers. Only after the organization rethinks what it should be doing, does it go on to decide how best to do it. The central aim being to achieve dramatic improvements in critical performance measures, such as cost, quality, service, and speed [89].

Re-engineering recognizes that an organization's business processes are usually fragmented into sub-processes and tasks that are carried out by several specialized functional areas within the organization. Often, no one is responsible for the overall performance of the entire process. Re-engineering maintains that optimizing the performance of sub-processes can result in some benefits, but cannot yield dramatic improvements if the process itself is fundamentally inefficient and outmoded. For that reason, re-engineering focuses on re-designing the process as a whole in order to achieve the greatest possible benefits to the organization and their customers. This drive for realizing dramatic improvements by fundamentally re-thinking how the organization's work should be done distinguishes re-engineering from earlier process improvement efforts that focused on specific functional improvements [89].

As the outside environment in which an organisation operates is subject to continual change it is not sufficient to simply perform a single iteration of a BPR. Therefore current BPR methodologies are instead focused on being continually repeated in the form of the Business Process Re-engineering Life

Cycle (PRLC) [123], illustrated in Figure 8.1. In this case the change performed during each cycle is intended to be less radical than proposed in the original formulation of the BPR framework.

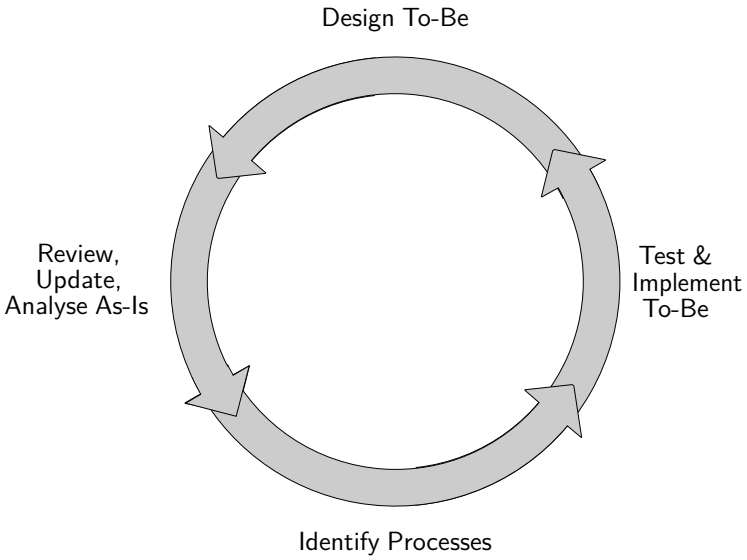


Figure 8.1: *PRLC: The business process re-engineering life cycle. (Source: [123])*

In Algorithm 4 the main steps of the process re-engineering life cycle are:

1. **Design To-Be**: In line with the overall BPR approach the first step is to envision the desired improved situation.
2. **Review, Update, Analyse As-Is**: This step is concerned with analysing the current situation and identifying the gap between the desired and the current situation.
3. **Identify Processes**: This step is concerned with identifying the processes that require modification in order to achieve the desired *To-Be* situation.
4. **Test & Implement To-Be**: The last step is concerned with testing and implementing the modifications to the identified processes in order to achieve the desired *To-Be* situation.

The BPR approach seeks to improve business processes and is a strictly manual process, where the term *improvement* implies a qualitative approach of developing an existing business process to a better version. In this chapter, the analysis framework for Core Business Process Model and Notation (BPMN) models established in Chapter 5 is extended to allow for *optimization* of Core BPMN

models, here *optimization* refers to the automated improvement of business processes using quantitative measures to evaluate the degree of improvement. This approach greatly simplifies implementing a process re-engineering life cycle as the automated optimisation can help guide the design of new processes by suggesting new improved designs of an existing process.

A survey by Mansar and Reijers [181] conducted in 2003 of 24 UK and Dutch companies which were BPR practitioners leads to the suggestion of a set of best practices to be employed when implementing BPR in terms of trying to deliver a process design that is in some sense superior to the existing one. These best practices, shown in Table 8.1, motivate the design of the automated analysis framework developed here.

The optimisation approach presented here allows for the optimisation of business processes where the optimisation objectives are broadly defined. Specifically, *optimisation goals* are defined using real-valued Probabilistic Computation Tree Logic (PCTL) queries and model checking is employed to construct a weighted fitness score for a given business process with respect to the optimisation goals. The core functional requirements for an optimised variant of a business process are similarly defined using PCTL queries, but with the difference that they are required to return boolean values. Central to this approach is a convenient representation and set of functions that modify Core BPMN models using to introduce parallelism or perform resequencing, as suggested in Table 8.1. This representation and associated modification functions are then used in an evolutionary algorithm which produces an improved business process in the same BPMN formalism as the input business model. The overall approach is shown in Figure 8.2.

It should be noted that the improved model generated by the approach shown in Figure 8.2 can then itself be used as the basis for further optimisation as illustrated by the dashed line. In this case termination of the optimisation process depends on a choice of how many generations of derived models ought to be explored. Further, it should be noted that this approach can be parallellised more than is illustrated in Figure 8.2 such that variant generation produces a range of models which are checked in parallel for each iteration of the algorithm. Finally, it should be noted that the variant generation step can be tuned to adjust the relationship between the amount of *crossover* and *mutation* performed.

Best Practice	Usage	Definition
Task Elimination	94%	Eliminate unnecessary tasks from a business
Integral Business Technology	94%	Try to elevate physical constraints in a business process by applying new technology
Task Composition	89%	Combine small tasks into composite tasks and divide large tasks into workable smaller tasks
Parallelism	88%	Consider whether tasks may be executed in parallel
Specialist-generalist	88%	Consider to make resources more specialised or more generalist
Resequencing	88%	Move tasks to more appropriate positions in the sequence
Integration	76%	Consider integration with a business process of the customer or a supplier
Empower	76%	Give workers most of the decision-making authority and reduce middle management
Numerical involvement	76%	Minimise the number of departments, groups and persons involved in a business process
Order assignment	53%	Let workers perform as many steps as possible for single orders

Table 8.1: *Definition and level of usage of BPR best practices amongst practitioners (Source [181])*

8.1.1 Related Work

Compared with the large number of proposed business process modelling techniques and manual improvement methodology approaches a study by Vergidis et. al. [278] concludes that business process optimization, in the automated sense, has received relatively little coverage [278]

Awad et. al. [34] introduced a semi-automated approach to synthesizing process templates out of compliance requirements expressed in *linear temporal logic* (LTL). Based upon a tableau proof procedure for temporal logic, this work

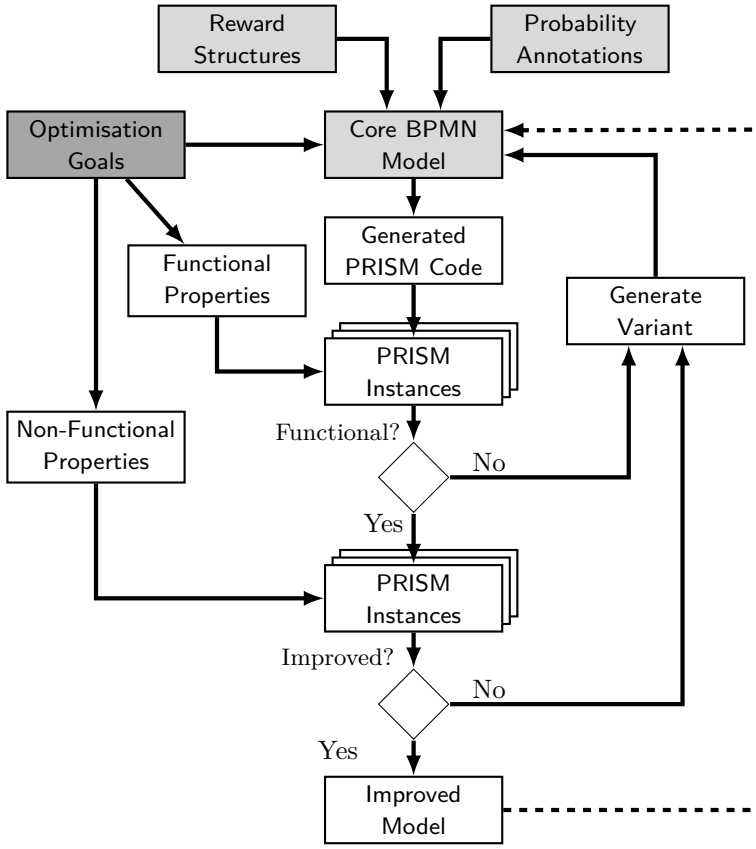


Figure 8.2: *The overall approach taken to automated business process optimisation.*

generates process models that are synthesized from compliance rules such as the Sarbanes-Oxley Act. Similar work by Goedertier and Vanthienen [117] developed an algorithm to generate compliant sequence-flow-based process models that can be used in business process design. The logic behind the obligations and permissions is made explicit in the form of temporal deontic assignments. In both cases the generated processes cannot be directly used for implementing a business process, and serve merely as templates for a final process design.

A related approach to process model synthesis is work by Pešić et. al. [219] who proposed a more declarative approach. Their *DecSerFlow* language, based on LTL rather than a traditional imperative process modelling language, builds models by specifying *what* should be done without specifying *how* it should be done. This is done by exploiting the fact that for every LTL formula a

Büchi automaton can be generated. However, these automata are not readily understood by the business community and additional work remains to be done to translate these automata back into business process models.

A data mining approach has been proposed by van der Aalst et. al. [26] with the aim of deriving business models from workflow logs containing information about a business process as it is being executed. Their method, termed *workflow mining*, has a usably low complexity bound, but suffers from the need to extensively instrument an existing business process so as to be able to generate workflow logs. Further, problems of noisy log data and deriving more complex properties than task ordering remain to be solved [21]. Finally, the resultant business model takes the form of a *Petri-net*, a formalism also not commonly employed by the business community.

In [22], van der Aalst and van der Hee present another approach to automating BPR approach, where business processes are modelled as *high-level Petri-net* and formal analysis techniques are applied to determine a number of performance properties with the aim of optimising resource allocation. While this work presents a powerful technique to allocate resources it does not restructure the Petri-net model itself. Instead, the step of arriving at a "To-be" model from an "As-is" model, shown in Figure 8.1, requires creative input from a business process designer who must imagine how a model can be altered to further improve it. Further, this work does not incorporate stochasticity, limiting its applicability to real-world business processes.

An approach by Jaeger and Prakashby [147] does allow for automatic restructuring of a business processes models where quantitative goals are used to guide the search for improved models. Analysis of individual models is done by means of simulation, and how quantitative properties are evaluated must be manually defined. The search for improved business processes employs an artificial intelligence inspired approach where a number of heuristic functions are used to make a guess at which modifications from a predefined set are likely to produce improvements. While this work does not incorporate stochasticity, it does seem to scale quite well and is able to output improved models, described using a specially defined annotated flowchart notation that is also used as input.

A unique approach is taken by Kamrani et. al. [150] which allows optimizing a BPMN business process model with the objective of finding the most beneficial assignment of tasks to agents, without modifying the structure of the process itself. However, only basic resequencing of the model is performed, the optimisation goals are limited, and the measure of model suitability is only approximately determined by simulation.

An approach which makes use of a evolutionary algorithm approach is that developed by Medeiros [184], where data mining event logs of executing business process are used as input to a genetic algorithm which constructs *workflow-net* models which are both complete (can reproduce all the behaviour in the log) and precise (do not allow for extra behaviour that cannot be derived from the event log). A key innovation of this work is the use of an adjacency style matrix representation of the structure of the workflow net which is similar to what is employed in this chapter. A key distinction between the analysis of this chapter and the work of Medeiros is that the Core BPMN input language used here allows for stochastic behaviour and the focus for this analysis is on design-time optimisation of processes where event log data is not available.

8.2 Optimisation of Core BPMN models

In this chapter, optimisation of Core BPMN Business Process Diagrams (BPDs) is inspired by the PRLC approach, shown in Figure 8.1, where the core concept of iteratively improving processes is intended to performed in an automated fashion. The crucial step of how to modify a process at each iteration is in turn motivated by Table 8.1, where specifically *resequencing* and *parallelisation* are chosen as the BPR practices which are most feasible for automated application. The overall approach is shown in Figure 8.2 and is intended to mimic the process of biological evolution. It employs a genotype-style representation of the optimisation problem, variation and selection operators, a fitness function, and can be formally classed as a evolutionary algorithm [118]. The initial seed from which evolutionary development can be performed is assumed to be a model of an existing initial manually constructed model of a new process.

The presentation of this optimisation approach will begin by presenting the various functions which are employed in the evolutionary algorithm, which is then presented in Section 8.2.6.

8.2.1 Model Checking Functions

The following two definitions express the application of model checking for the specified path formulae of a Core BPMN BPD for a specific query target value, and for an evaluation of an unspecified reward or probability bound by means of

the ? notation which returns the calculated probability or reward value. Formally, the evaluation of PCTL queries $f, q \in \mathbf{Q}$ with a specified target value (f case) or quantitatively evaluated (q case) is defined as:

Definition 8.1 (Quantitative PCTL query evaluation)
Given a well-formed BPD and a PCTL query q the function $\mathsf{MC}_? : \mathbf{BPD} \times \mathbf{Q} \rightarrow \mathbb{R}^+$ is defined as the quantitative model checking result of BPD for q, such that:*

$$\mathsf{MC}_?(BPD, q) \in \begin{cases} [0,1] & \text{if } q \text{ contains } \mathsf{P} \\ [0,\infty[& \text{if } q \text{ contains } \mathsf{R} \end{cases} \tag{8.1}$$

Definition 8.2 (Specific PCTL* query evaluation)
Given a well-formed BPD, a PCTL query f and a specific query value n, the function $\mathsf{MC}_\mathsf{n} : \mathbf{BPD} \times \mathbf{Q} \rightarrow \{true, false\}$ performs model checking of the BPD to determine if f is satisfied.*

8.2.2 Optimisation Goals

To make it possible to combine multiple weighted objectives, and to have sets of optimisation goals in which both rewards and event probabilities can be expressed, an individual optimisation goal is defined as follows:

Definition 8.3 (Optimisation Goal Tuple)
A goal is a tuple $G = (t, w, q)$, where $t \in \{min, max\}$ denotes if the goal is to be minimised or maximised, $w \in \mathbb{R}^+$ is a positive real-valued weight denoting the relative importance of the goal, and q is a PRISM PCTL query.*

In practice, it is frequently desirable to optimise a business process with regard to multiple quantitative properties and the set $\mathbf{G}$ is used to denote a set of optimisation goal tuples. The quantitative PCTL* query evaluation function defined in Definition 8.1 is employed in fitness scoring. For a set of optimisation goal tuples $\mathbf{G}$, the relative improvement of a new BPD BPD' compared to the existing BPD BPD is evaluated by using following function:

Definition 8.4 (Optimisation Goals Scoring)
Given a well-formed BPD and a set $\mathbf{G}$ of optimisation goal tuples, the function $\mathsf{Score} : \mathbf{BPD}^2 \times \mathbf{G} \to \mathbb{R}$ *is defined as the model checking result of BPD for q, such that:*

$$\mathsf{Score}(BPD, BPD', \mathbf{G}) = \sum_{(t,w,q)\in\mathbf{G}} \begin{cases} w\left(\mathsf{MC}_?(BPD', q) - \mathsf{MC}_?(BPD, q)\right) & \text{if } \mathsf{type}(q) = \mathsf{P} \wedge t = max \\ w\left(\mathsf{MC}_?(BPD, q) - \mathsf{MC}_?(BPD', q)\right) & \text{if } \mathsf{type}(q) = \mathsf{P} \wedge t = min \\ w\left(\frac{\mathsf{MC}_?(BPD', q) - \mathsf{MC}_?(BPD, q)}{\mathsf{MC}_?(BPD, q)}\right) & \text{if } \mathsf{type}(q) = \mathsf{R} \wedge t = max \\ w\left(\frac{\mathsf{MC}_?(BPD, q) - \mathsf{MC}_?(BPD', q)}{\mathsf{MC}_?(BPD, q)}\right) & \text{if } \mathsf{type}(q) = \mathsf{R} \wedge t = min \end{cases} \tag{8.2}$$

where $\mathsf{type} : \mathbf{Q} \to \{\mathsf{R}, \mathsf{P}\}$ *determines if a PCTL* query $q \in \mathbf{Q}$ is a reward or probability based query.*

8.2.3 Functional Requirements

Functional requirements allow the expression of properties which must hold for any future business process BPD' derived from a BPD. The set of functional requirements for an optimised BPD is denoted by $\mathbf{F}$. Like optimization goals, functional requirements will be defined using PCTL* formulae. In this case, however, it is required that probabilities or reward values within the query are explicitly defined, such that the return value of the query is a boolean variable as defined in Definition 8.2.

Definition 8.5 (Functional requirements check)
Given a well-formed BPD and a functional requirement f the function $\mathsf{FCheck} : \mathbf{BPD} \times \mathbf{F} \to \{true, false\}$ *determines the BPD's compliance with $\mathbf{F}$ as:*

$$\mathsf{FCheck}(BPD, \mathbf{F}) = \bigwedge_{f\in\mathbf{F}} \mathsf{MC}_\mathsf{n}(f) \tag{8.3}$$

Note that for *must* or *must not* criteria, P operators are defined with explicitly defined probabilities of 1 or 0 respectively. This conjunctive normal form of functional requirements can be efficiently checked in parallel with FCheck returning false as soon as a requirement is encountered that MC_n determines to be false.

8.2.4 Selection

A key step in development of an evolutionary algorithm is the selection of members of a current generation used to derive the next generation. In Definition 8.6, stochastic sampling with limited replacement is employed. In essence, each member of a current generation is mapped to a contiguous segment of a line, such that each individual's segment is proportional in length to its fitness. A random number is generated and an individual A whose segment spans the random number is selected. This process is repeated to obtain a partner with the restriction that if A is selected a new sample is chosen.

Definition 8.6 (Selection Function)
Given a set of BPDs **BPD** *with fitness scores given by* $f : \mathbf{BPD} \rightarrow \mathbb{R}$*, a pair of BPDs* (BPD_A, BPD_B) *is selected such that* $BPD_A \neq BPD_B$ *with the following probability:*

$$P(BPD_A \in \mathbf{BPD}) = \frac{|f(BPD_A)|}{\sum_{BPD \in \mathbf{BPD}} |f(BPD)|} \tag{8.4}$$

8.2.5 Variant Generation

When generating variants, the traditional evolutionary algorithm approach of constructing a separate genotype representation upon which to perform modification of a BPD is employed. This approach allows the genotype structure to closely reflect the phenotype structure, and encodings with this property are believed to make the evolutionary algorithm more robust (i.e. reduce the probability of fatal mutations), and also improve a system's capacity for adaptive evolution [118].

An adjacency matrix style representation $MBPD$ for this genotype is used, where the vectors $v_{i,j}$ store the reward structures associated with nodes $n \in \mathbf{N}$ of the BPD. The phenotype is simply the BPD that is derived from this representation.

$$MBPD = \begin{array}{c} \mathbf{N} \\ n_1 \\ n_2 \\ \vdots \\ n_i \end{array} \begin{array}{c} \begin{array}{cccc} n_1 & n_2 & \cdots & n_j \end{array} \\ \begin{pmatrix} 0 & v_{1,2} & \cdots & v_{1,j} \\ v_{2,1} & 0 & \cdots & v_{2,j} \\ \vdots & \vdots & \ddots & \vdots \\ v_{i,1} & v_{i,2} & \cdots & 0 \end{pmatrix} \end{array} \tag{8.5}$$

where

$$v_{i,j} = \begin{cases} (p_{i,j}, R_{1,i}, \cdots) & \text{if } n_i \text{ links } n_j \text{ with probability } p_{i,j} \\ (0, 0, \cdots) & \text{otherwise} \end{cases} \tag{8.6}$$

This representation is somewhat similar to the approach of Medeiros [184], with the matrix representing the causal relationship between elements in a business process. However, the vectors also record quantitative data associated with each state.

The function BPD2MBPD : **BPD** $\rightarrow$ **MBPD** maps a BPD to its adjacency matrix style representation, and the MBPD2BPD : **MBPD** $\rightarrow$ **BPD** maps a **MBPD** to a BPD. This is done using the well-established technique outlined in [80] with complexity $O(n)$ where n is the number of nodes in BPD.

8.2.5.1 Crossover

Crossover is a genetic operator that aims at recombining existing material in a current population.

Instead of creating offspring by swapping information from two parents based upon one or more points in a linear structure as in done in genetic algorithms [118], a rectangular section of the matrix structure is selected at random. The approach to crossover follows naturally from the structure of the genotype representation presented in eq. (8.5). The result of the crossover is then created by using information from inside a selection rectangle of one parent, and outside the selection rectangle of the other parent, as illustrated in Figure 8.3.

Various possible optimisations when implementing Definition 8.7 are discussed in Section 8.4.

Definition 8.7 (Crossover Operator)
Given a pair of BPDs $(A, B) \in$ **BPD** *with defined MBPD representations, crossover is defined by* Cross : (**MBPD** $\times$ **MBPD**) $\rightarrow$ **MBPD** *where* $C =$ Cross(A, B) *such that:*

$$C[r_i - r_j; c_n - c_m] = B[r_i - r_j; c_n - c_m]$$

and

$$C \backslash [r_i - r_j; c_n - c_m] = A \backslash [r_i - r_j; c_n - c_m]$$

where the sub-matrix indices, r_i, r_j, c_n, c_m *are randomly chosen between* $[0, min(dim(A), dim(B))]$ *such that* $(r_j < r_i) \wedge (c_m < c_n)$.

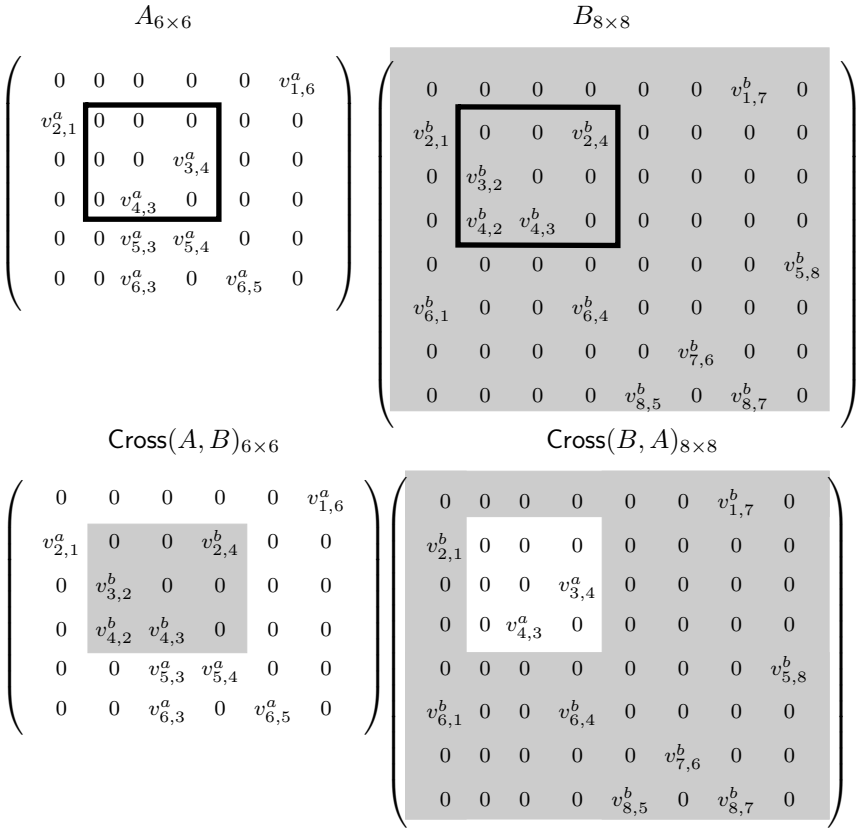


Figure 8.3: *Illustration of the crossover operator applied to MBPD representation of a business process.*

8.2.5.2 Mutation

The mutation operator aims at inserting new material in the current population. Mutation is also an operator which is applied to the $MBPD$ representation of a BPD and is intended to complement the crossover operator by injecting small local changes to a BPD. Mutation is defined so as to allow for considerable variation of a source BPD.

Definition 8.8 (Mutation Operator)
Given a BPD, with a defined $MBPD$ representation and mutation rates $r_{resequence}, r_{parallelize} \in [0,1]$ perform mutation $\mathsf{Mutate} : \mathbf{MBPD}\times \rightarrow \mathbf{MBPD}$ *such that for each row at index m in the $MBPDs$ if* $\mathsf{Type}(n_i) = \mathbf{T}$*:*

1. *With probability* $r_{resequence}$ *swap the column of the non-zero vector* v_{m,j_1} *to a new position* $j_2 \neq j_1$.
2. *With probability* $r_{parallelize}$ *add rows* $r_m - 1$ *and* $r_m + 1$ *to the MBPD and add vectors* v_{m-1,j_1} *and* v_{m+1,j_2} *in the columns* j_1 *and* j_2 *such that for the element* $v_{m,j}$ *the following sequence relations* $n_{m-1}\mathcal{S}n_m$ *and* $n\mathcal{S}n_{m+1}$ *hold.*

Definition 8.8 allows mutations to have two effects on a BPD:

1. **Resequencing:** Illustrated in Figure 8.4, this modification alters the BPD element $\mathcal{S}$ which defines sequence flows. Specifically, it alters the relation between two nodes in the sequence flow (e.g. A and B in Figure 8.4(a)), replacing the destination node with a different node (Figure 8.4(b)) and reconnecting any excluded nodes to follow after the resequencing (Figure 8.4(c)). In effect, randomly reordering the sequencing of a number of tasks in the BPD.

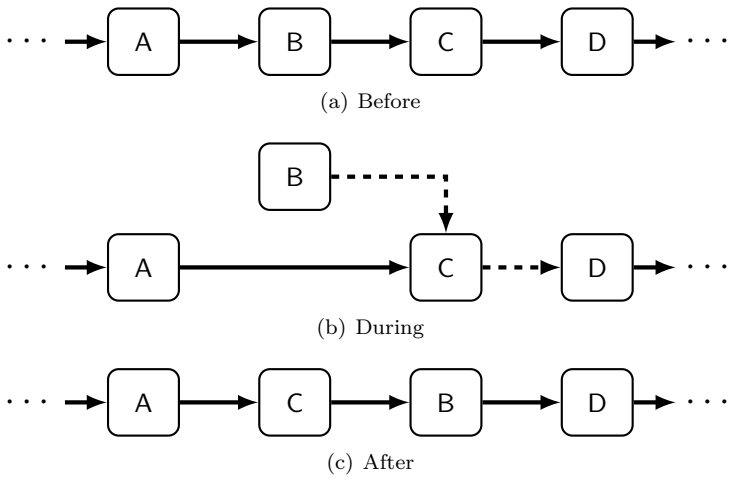


Figure 8.4: *Illustration of application of the resequencing operator.*

2. **Parallelisation:** This modification is illustrated in Figure 8.5 and functions by injecting pairs of parallel merge and fork gateways. These can be injected at any point other than at start and end elements (e.g. between A and D in Figure 8.5(a)), and the nodes between the injected gateways are initially all assigned to one of the parallel paths (e.g. Figure 8.5(b)). Note that when this is combined with the resequencing operator both parallel branches will eventually contain nodes.

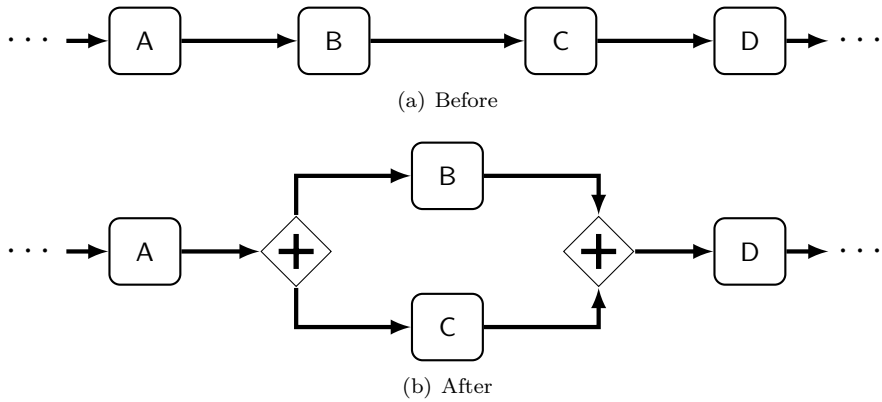


Figure 8.5: *Illustration of application of the parallelisation operator.*

It is tempting to consider a more refined version of the parallelisation operator which avoids performing parallelisation in cases when it obviously will produce an invalid variant, i.e. their is a constraint which explicitly forbids the parallelisation of two tasks. However, in practice no refinement has been found which does not involve performing an amount of work that is broadly equal to the burden of simply model checking to check if the variant is valid.

8.2.6 Optimisation Algorithm

The approach in this thesis to the optimisation of business processes is to take an existing business process, modelled as a Core BPMN BPD and search through possible iterative modifications of this BPD to arrive at an improved version in the fashion shown in Figure 8.2. Algorithm 5 uses the key functions described above to perform optimisation of a BPD and, by performing modifications directly on the BPD, ensures that the final improved process is also a BPD and requires no special interpretation by end users.

The influence of the various constants employed in Algorithm 5 is discussed separately in Section 8.4.

In Algorithm 5, an initial population of variants of the input BPD of size pop_size is generated by means of a *while* loop in lines 2-7. Specifically, each variant BPD' is constructed by applying the MBPD2BPD function to the Mutate function operating on the $MBPD$ representation of the original BPD with a chosen mutation probability rate of $r_{mutation\_initial}$ (line 3). Due to the computational expense of performing quantitative model checking of a BPD,

Algorithm 5: BPD Optimisation

Input: A Core BPMN BPD with Rewards, a set of optimisation goal tuples $\mathbf{G}$, a set of functional requirements $\mathbf{F}$, and a generation limit and population size $gen\_limit, \text{pop\_size} \in \mathbb{N}^+$

Output: A BPD optimised with regard to $\mathbf{G}$ and not violating $\mathbf{F}$.

```
1  i ← 0
2  while (i < pop_size) do                    //Generate initial population
3      BPD' ← MBPD2BPD(Mutate(BPD2MBPD(BPD), r_mutatuion_initial))
4      if WellFormed(BPD') ∧ FCheck(BPD', F) then
5          Variants[i] ← BPD'
6          Fitness[i] ← Score(BPD, BPD', G)
7          i++
8  generation ← 0
9  while (generation ≤ gen_limit) do          //Evolve population
10     i ← 0
11     while (i < pop_size) do                //Generate children
12         (BPD_A, BPD_B) ← SelectPair(Variants[i], Fitness[i])
13         BPD' ← Cross(BPD2MBPD(BPD_A), BPD2MBPD(BPD_B))
14         BPD' ← Mutate(BPD', r_resequence, r_parallelize)
15         BPD' ← MBPD2BPD(BPD')
16         if (WellFormed(BPD') ∧ FCheck(BPD', F)) then
17             Variants[i] ← BPD'
18             Fitness[i] ← Score(BPD, BPD', G)
19             i++
20     BPD ← ChooseHighest(Variants[], Fitness[])
21     generation++
22 return BPD
```

some variants are excluded before evaluating their quantitative properties. This filtering is performed in line 4 where the function WellFormed determines if the structural semantic rules of Section 3.3.2 hold for BPD' and function FCheck (see Definition 8.5) checks if BPD' conforms with a set of functional requirements $\mathbf{F}$. Subsequently BPD' is added to an array of variants (line 5) and its corresponding fitness score as determined by the function Score (see Definition 8.4) is stored in a separate fitness array at the same index point i in line 6. Finally the array index is incremented before the generation of the next variant in line 7.

In line 8 a *generation* counter is initialised. Subsequently in lines 9 -21, *the evolution* of the initial population takes place within a *while* loop for a number of generations determined by the generation limit *gen_limit* which is an input to the algorithm. First in line 10, a counter of the current population size of a generation is initialised to 0.

Next a new child population, of size *pop_size*, is generated by means of a *while* loop in lines 11 -19. This is done, in line 12, by selecting a pair of BPDs from using the function SelectPair from elements of the array of variants of the previous generation in a fashion that is proportional to their associated fitness scores (see Definition 8.6) (note in the first iteration these variants will be members of the initial population). This pair is used to generate a new variant BPD' using the crossover operator Cross applied to $MBPD$ representations of BPD' (line 13). Next, in line 14, the function Mutate performs a number of alterations of BPD', currently containing an $MBPD$ representation, dictated by the mutation rates $r_{resequence}$ and $r_{parallelize}$. At this point, in line 15, the $MBPD$ representation BPD' is mapped back to a standard BPD representation.

If a variant proves to obey the structural semantic rules of Section 3.3.2 and meets functional requirements **F** (line 16), it becomes part of the next generation by being added to the variant array (line 17) where its associated fitness score is stored in the associated fitness array (line 18) and the array population index is incremented (line 19).

Having generated a new child population, the function ChooseHighest selects the highest fitness scoring member of the final generation with regard to the optimisation goals **G** and stores it as the current BPD (line 20). In line 19 the *generation* counter is incremented, before the next iteration of the generation loop.

Having completed execution of Algorithm 5 the BPD BPD will contain the most optimal, functionally correct variant of the original input BPD that could be found, within the search parameters. Note that Algorithm 5 may not terminate as generation of populations may never produce a child that is both functionally correct and obeys the Core BPMN structural semantic rules. In this case an implementation should seek to terminate variant generation and present the result found so far. Alternatively a new initial population can be generated and crossover performed between this population and the best found so far. A number of specific implementation strategies are discussed in Section 8.4.

The worst case execution time, in the case when Algorithm 5 terminates, is $O(nm)$ where n and m are *gen_limit* and pop_size. Note that each execution involves performing PCTL model checking, which in turn is linear in the size of the PCTL formula $|\phi|$ and polynomial in the size of the state space $|S|$ (see Appendix A.7).

8.3 Optimisation Example

To illustrate an application of this method, an example of a simple business process inspired by hospital operations of the type described in Section 1.1, where patients are treated with chemotherapy drugs, is used. Fig. 8.6 is an example of such a process which is annotated with rewards and information about its stochastic behaviour. This naively-designed business process consists of two processes: *Doctor*, modelling the actions of a doctor, and *PharmacyRobot*, modelling the actions of a robot which prepares drugs for use by the doctor.

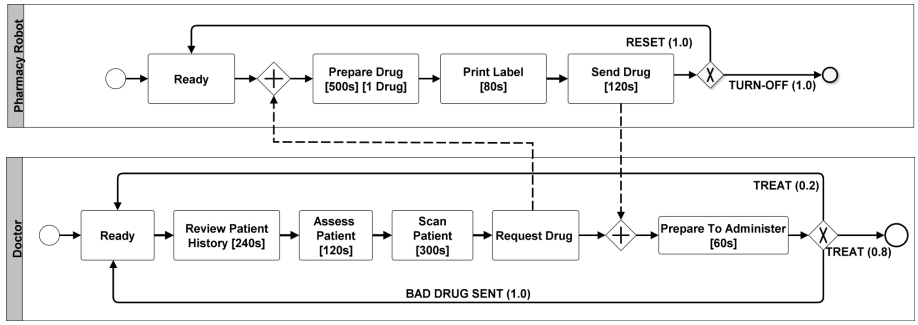


Figure 8.6: *BPD model of a doctor and pharmacy robot system before optimisation.*

In this model, when the *Doctor* process is in the ready state, a patient is received, and the doctor then proceeds to perform actions in a linear sequence. Specifically, the doctor begins by examining the patient's medical records which takes 240 seconds. He then questions and assesses the patient (120 seconds) and then scans the patient (300 seconds). In this simple example, the doctor always proceeds to treat the patient by requesting a chemotherapy drug from the *PharmacyRobot* process.

The *PharmacyRobot* process can progress beyond its ready state when an order is received. At this point it proceeds to prepare a drug, an action which has two associated reward structures, capturing the time taken (500 seconds) and the amount of drugs used (1 drug). It then prints a tracking label to be placed upon

the order (80 seconds) and sends the drug (120 seconds). After having sent off the drug order, the robot makes a choice between resetting to its ready state or turning off and going offline. In terms of the semantics of stochastic Core BPMN this means that the *PharmacyRobot* process makes a non-deterministic choice between these options, where each choice has an associated probabilistic value with a probability of 1.0.

Once the *Doctor* process receives the drug, the next step is to prepare to administer it (60 seconds). After preparing the drug the *Doctor* process makes a non-deterministic decision whether to treat the patient with the drug, or whether the Doctor has deemed that a *bad* (incorrect or unsafe) dose of drug has been sent and resets the process. In the case when the *Doctor* process decides the drug is ok and chooses to treat the patient there are two probabilistic outcomes of this action: with a probability of 0.8 the treatment is successful and the *Doctor* process reaches its end state. However, in the case where treatment is the decided cause of action, there is a probability of 0.2 that the treatment is not effective and, in this simple example, the application of the same drug is eventually repeated.

8.3.1 Example Optimisation Goals

Inspired by requirements in hospital operations of the type described in Section 1.1, the following set of optimisation goals and functional requirements are a reduced but representative set, of the type of goals and requirements placed on medical workflows.

For the business process described in Figure 8.6 it is desirable to see an improvement in the time taken for a doctor to complete treatment of a patient. Further, it would also be desirable that the rate of drug consumption and the consequent probability, given a specific drug stock size, of running out of the drug is kept as low as possible. These requirements **G** are expressed, using goals from Definition 8.3 as:

$$\mathbf{G} = \{(min, 1, \mathsf{R}^{\mathtt{time}}_{\mathsf{min}}[\mathsf{F}\ SendHome]), (min, 1, \mathsf{P}_{\mathsf{min}}[\mathsf{F}\ DrugExhausted])\} \tag{8.7}$$

In addition to the optimisation goals, a number of functional requirements, **F**, exist for this process:

1. The review of the patient's medical history should take place before the *Doctor* assesses the patient:

$$\mathsf{P}_{\mathsf{min}} = 1\,[\ \mathit{review}\ \mathsf{U}\ \mathit{assess}\ ]$$

2. All analysis of the patient, assessment, scanning, history review (in any order), must take place before a drug is administered:
$$\mathsf{P}_{\mathsf{min}} = 1\,[\mathsf{F}(\mathit{assess}\ \wedge\ \mathit{scan}\ \wedge\ \mathit{history}\ )\ \mathsf{U}\ \mathit{admin}\ ]$$
3. The *Doctor* cannot administer the drug before he has requested it.
$$\mathsf{P}_{\mathsf{max}} = 0\,[\ \mathit{admin}\ \mathsf{U}\ \mathit{request}\ ]$$
4. The *PharmacyRobot* must ensure that a drug has been prepared and labelled (in any order) before it is sent to the doctor.
$$\mathsf{P}_{\mathsf{min}} = 1\,[\mathsf{F}(\ \mathit{prepare}\ \wedge\ \mathit{label}\ )\ \mathsf{U}\ \mathit{send}\ ]$$
5. When the doctor determines that a bad dose of drug has been received, he must immediately request a new dose.
$$\mathsf{P}_{\mathsf{min}} = 1\,[\mathsf{F}(\ \mathit{bad\_drug}\ (\mathsf{X}\ \mathit{request}\ ))]$$

Note that the final functional requirement (Item 5) is not currently satisfied by the initial *BPD* shown in Figure 8.6.

8.3.2 Example Optimisation Outcomes

Figure 8.7 illustrates one possible outcome of applying Algorithm 5 to the *BPD* shown in Figure 8.6. Specifically, this is the outcome of 28 generational improvements of population size 500 of the variants per generation. In the case of this example, the rates $r_{resequence}$ and $r_{parallelize}$ are set so that the Mutate function ensures that considerably more resequencing modifications are performed than parallelisation modifications. The exact choice of these rates was found through experimentation with the values of $r_{resequence} \in [0.25, 0.65]$ and values of $r_{parallelize} \in [0.1, 0.2]$ frequently leading to convergence on a process exhibiting some degree of improvement in less than 30 generations.

In this run of the optimisation method, two opportunities to parallelize actions are identified. Within the *PharmacyRobot* process, the drug can be prepared and the label printed to be placed on the order at the same time. In the *Doctor* process, it is possible to scan the patient while simultaneously reviewing the medical record and then doing an assessment of the patient. In both cases this saves time, as when performing actions in parallel only the path with the slowest behaviour is counted towards the parallel section's contribution to the reward value. Further, note that the new functional requirement (Item 5) is now satisfied and receiving a *bad_dose* of drug immediately leads to a request for a new dose. It should be noted that doing tasks in parallel might be too demanding for the *Doctor* process if it only consists of a single human doctor, but even in this case it can suggest new designs for the process and technology, such as an automated medical scanner, could be used to realise it.

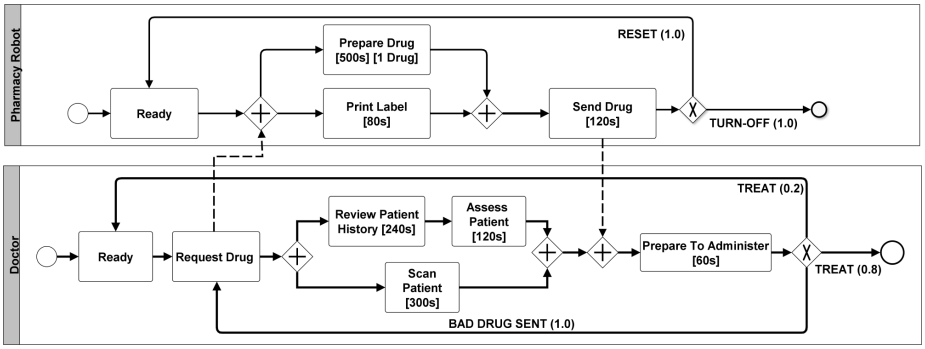


Figure 8.7: *Post optimisations BPD model of a doctor and pharmacy robot system after* 28 *generations with a population size* 500.

In this simplified example, the doctor always orders a drug to treat a patient, but the doctor must wait while the Pharmacy Robot performs its operations and then delivers the drug. As the drug will inevitably be needed, and only the administration of the drug needs to be done after the patient has been examined, it is within the functional requirements for the process, and results in a considerable time saving, to order the drug immediately before even examining the patient. This ensures that there will be no delay imposed on the Doctor process by the actions of the Pharmacy Robot process. While this optimisation seems somewhat unrealistic, it does not violate the functional requirements and is a natural consequence of the stated optimisation goals as it results in a significant reduction of the time taken for the execution of the business process. This highlights a limitation of the method in that the functional requirements for an optimal process must be carefully specified.

8.4 Evaluation of the Optimisation Method

This section reflects on the pros and cons regarding the proposed optimisation approach for a qualitative perspective. The performance and scalability of this approach is explored in Section 9.3.5.

The foundations of the synthesis of concurrent programs under quantitative constraints have been pursued by Černý et. al. [68], resulting in a proof that the complexity of this category of synthesis problems lies in the **NEXP**-complete complexity class. Hence, in the general case, they can be solved in time $O(2^{p(n)})$ for some polynomial $p(n)$. While this bound also applies to the method described

here, evolutionary algorithms can often find near optimal solutions considerably faster. However, the fundamentally stochastic nature of evolutionary algorithms makes predicting the time taken for each optimisation search challenging and several runs of the algorithm with the same tuning properties may have convergence times that vary by several orders of magnitude.

The focus of efforts to obtain more reliable performance from the optimisation approach is to employ *parallelisation* of individual model checking runs. In particular the generation limit *gen_limit* and population size of *pop_size* dictate the average run time. Tuning these values to closely match the available computational resources in terms of parallel processing units available provides significant speed up if all processing units can be kept busy. Experimentally extending child generation runs in Algorithm 5 to terminate when, a threshold fraction of parallel processing units become idle leads to overall reductions in run time at the expense of greater variance in convergence time. This approach leads to generally better results, both in terms of convergence and degree of improvement, than an alternative where, as fitness checking steps are completed (evaluation of the score function), results are compared, and if significant improvement is not discovered, the pathologically slow searches are terminated. This seems to be due to the fact that while obviously incorrect models can quickly be checked, an individual long model checking run is a sign that no property violation has yet been found and may likely be a good candidate process for the next generation.

When performing crossover, ensuring basic structural restrictions provide better performance, by ensuring more variants are produced which obey the structural semantic rules. Computationally suitable restrictions are ensuring that the diagonals in the $MBPD$ must be zeros or that each task represented in the $MBPD$ has only one entry for each row or column. This provides a clear performance improvement in a practical implementation when applied to checking systems where model variant generation dominates the execution time. This is due to the fact that while crossover is less likely to produce a model that is not structurally sound than what is generated by mutation. However, it may well not meet the functional requirements which are checked via model checking.

In terms of the search of possible improved models performed by Algorithm 5, the population size, *pop_size*, determines the breadth of an optimisation search and the generation limit, *gen_limit*, the depth of search. Tuning these values is heavily dependent on the specific model being tackled and how far away an optimised solution is expected to lie from the current BPD. For problems which are already close optimal, increasing *pop_size* limit lead to faster convergence on high fitness score solutions, in fewer generations. Whereas in cases where a model is clearly far from optimal increasing the generation *gen_limit* increases the chance that an optimised variant that is significantly different from the original BPD will be found. However in the case of large values of *gen_limit*

a phenomenon known as *bloat* [192] occurs where after many generations of improvement the fitness score becomes largely static and the BPD starts growing at a rapid pace. Where typically the increased size is not accompanied by any corresponding increase in fitness.

Goal weights affect both the rate of convergence on a solution and, of course, the specific solution found. In theory, at generation ∞ one set of weights would give one specific solution from a set of equivalent solutions. [118]. However in practice, it seems best to experiment with a range of values around the desired goal weighting. Again the stochastic nature of evolutionary algorithms makes performing repeatable experiments in this area difficult.

The basic concept of evolutionary algorithms is that they are designed to simulate the natural process of survival of the fittest. Specifically, evolutionary algorithms generate new solutions to a problem in three ways. The *initialization* of the population is a source of new solutions. *Mutation* generates variations of existing solutions, and *crossover* blends solutions. When working well, crossover is a source of large innovations. However, an unavoidable problem with standard evolutionary algorithms is that they lose diversity rapidly, because after a number of generations, crossover happens mostly between elements of the same approximate type. The result is that most crossover is wasted effort. Mutation is therefore employed to help maintain diversity [118]. In the context of Algorithm 5, this practical principle of evolutionary algorithms was found to apply. Modifying the rates $r_{resequence}$ and $r_{parallelize}$ towards larger values (closer to 1) tended to lead to faster convergence towards a solution. However, too large values for these rates, typically $r_{resequence} \geq 0.85$ and $r_{parallelize} \geq 0.6$ prevent finding any improved process at all. This is due to the fact that the mutation operator must be tuned so as to successfully preserve diversity through successive generations by introducing a sufficient amount of new information into models. It must not produce too much diversity as this leads to a large number of BPDs which are invalid with respect to the structural semantic rules for Core BPMN. This reduces the approach virtually to simply trying random BPDs in the hope that they will provide a degree of improvement.

Fundamentally, choosing the correct *resequence* and *parallelize* rates is the mechanism by which "building blocks" of an improved solution are aggregated. From a theoretical perspective the effective variant of the Schema Theorem [192] that governs this work can be seen as conservatively parallelising and reordering sections of the BPD which contribute to optimising the chosen set of optimisation tuples $\mathbf{G}$. Further, when performing mutation, enforcing the structural semantic rules after each individual modification leads to an effect that seems to mirror the effect of reducing both *pop_size*, and *gen_limit*, leading to often to converging runs but which do not exhibit much improvement.

The structure of the source BPD also is of great significance when employing this optimisation approach. The number of reward annotations, for which there are specified optimisation goals, in the source BPD effects the tractability of the problem. If there are not enough rewards, initial improvements never get discovered, or a trivial solution (local maxima) dominates. Further, very few highly linear processes generally benefit from a higher $r_{resequence}$ rates than those with a large degree of branching. Further If the functional requirements make few explicit sequencing demands, increasing $r_{parallelize}$ tends to find higher fitness score solutions.

The need to specify functional requirements of an optimised BPD can be laborious and a complete set of requirements can become very large for a real-world model. However, the process of producing such a list has been found to be valuable to business practitioners as it focuses attention on what the actual purpose of the business process is. This is very much in line with the first step of the business process re-engineering life-cycle [89]. Fundamentally PCTL, even in the PCTL* variant, is not expressive enough to formalize all reasonable structural constraints, as formalizing the branching structure up to bi-similarity would require the full expressiveness of the µ-calculus.

The work of Pešić et. al. [219] employs Linear Temporal Logic (LTL) for functional requirements placed on web-services, and restricts the use of LTL to only represent sequences of single events for reasons of performance. The basic performance problems in the applying the optimisation approach presented here apply to the work of Pešić et. al. also. However, in this case when the functional requirements are specified in a fragment of PCTL* which restricted to only the probabilistic Computation Tree Logic (CTL) component an improvement is possible. The time complexity for PCTL model checking over an Markov Decision Process (MDP) is linear in the size of the PCTL formula and polynomial in the size of the models statespace $|S|$ [39]. However the overall complexity checking of LTL formulas is exponential in the LTL formula and polynomial in $|S|$ [39]. While this performance improvement of the optimisation approach presented here comes at the cost of expressivity of functional requirements, it does make this approach somewhat more feasible.

All in all this optimisation approach can still be considered experimental and requires considerable computing resources and considerable investment in terms of defining functional requirements to be readily applicable for dealing with real-world processes. A discussion of broader conclusions that can be drawn about these results and suggestions for future improvements will be in presented Chapter 10.

8.5 Chapter Summary

This chapter presents an advanced application of the developed framework for analysis of business processes which exploits the fact that model checking of business processes is relatively computationally inexpensive when using Probabilistic Symbolic Model Checker (PRISM) and can be parallellised for separate model checking problems. An evolutionary algorithm is developed to explore variants of a given business process for possible improvement, by means of mutation, crossover and selection operators. This algorithm is able to find optimised variants of Core BPMN processes, while observing a set of functional constraints which express properties which must hold for a final optimised process.

A small example is presented of the application of this method which is able to automatically reconfigure a business process under a set of functional requirements of weighted optimisation goals. This is used to motivate a discussion of the overall character of this approach.

Chapter 9

SBOAT

"Every program has at least one bug and can be shortened by at least one instruction - from which, by induction, it is evident that every program can be reduced to one instruction that does not work"
*(*Ken Arnold 1986*)*

Overview

This chapter describes the tool: Stochastic BPMN Optimisation Analysis Tool (SBOAT), which implements the unified specification and analysis framework presented in previous chapters. It allows a user to model Core Business Process Model and Notation (BPMN) processes, and annotate them with rewards, stochastic branching, and faults. The description of SBOAT covers the design of the GUI, central data structures and support for cloud compute resources, along with a number of practical issues to improve the performance of analyses presented. Using SBOAT, the extent to which the various analyses presented in this thesis are able to scale to tackle business processes of a range of sizes is explored.

The development with SBOAT and examples using SBOAT have been presented in previous work [3]–[5].

9.1 SBOAT

SBOAT is a software implementation of the main ideas in this thesis. It evolved from a previous tool: Stochastic BPMN Analysis Tool (SBAT) which only implemented the Core analysis presented in Chapter 5 (excluding scheduling analysis) and allowed for model construction via an eXtensible Markup Language (XML) based model description language for Core BPMN Business Process Diagrams (BPDs) loosely based around the XML Process Definition Language (XPDL) file format. SBAT was built on top of version 3.3.1 of the Probabilistic Symbolic Model Checker (PRISM) tool. The primary use of SBAT was as a pilot version for the development of SBOAT which implements the entire framework.

SBOAT, in its latest version (*0.45*) is built on top of PRISM version *4.1 dev-r7596* [217]. SBOAT implements a graphical user interface that should be readily familiar to users of Business Process Management (BPM) modelling tools. To enable the analysis to scale flexibly it employs cloud computing technologies that allow for applying significant parallel computational resources to the analyses when required.

Note that due to legal requirements and copyright infringements specified in a non-disclosure agreement with the industrial partner, the SBOAT tool cannot be made publicly available. Furthermore, the examples from the industrial partner of concrete medical workflows cannot be made available due to confidentiality arrangements between the industrial partner and its customers.

9.2 Overall SBOAT Design

An overview of the Architecture of SBOAT which enables the tool to perform analysis of Core BPMN models is shown in Figure 9.1. The main elements of this architecture are described in the following sections.

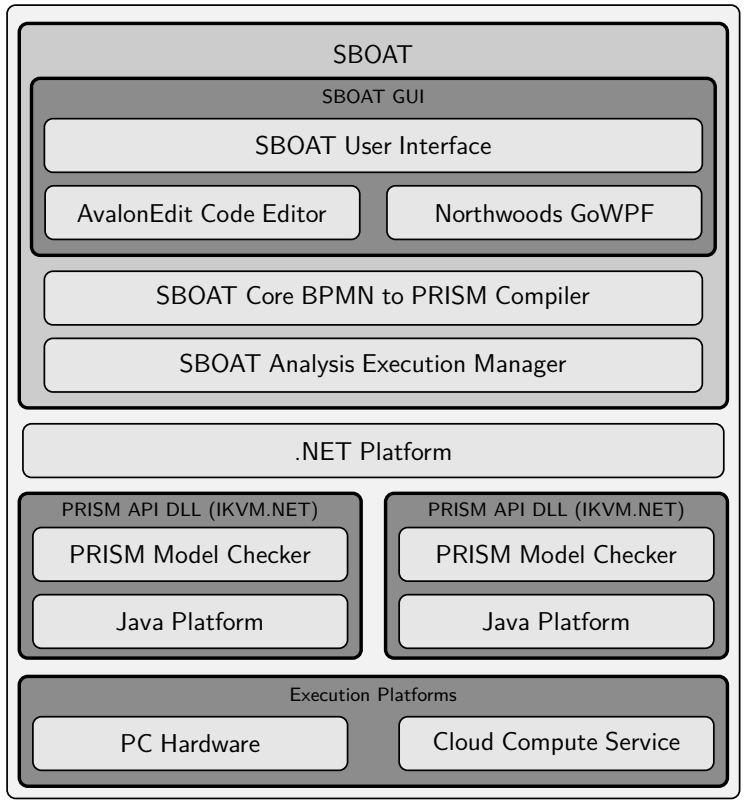


Figure 9.1: *Architecture of the SBOAT tool and environment.*

SBOAT is implemented in C# and employs IKVM.NET [107], a .NET implementation of a Java Virtual Machine, to be able to statically compile Java libraries, specifically the PRISM engine and API, into .NET assemblies. This allows for calling PRISM without the overhead involved in starting an instance of the Java virtual machine. A key motivation for employing C# for development was that the SBOAT would be able to be integrated into a larger tool-chain of the industrial partner. However, code profiling suggests that the memory allocation strategy of a .NET virtual machine leads to a slight speed improvement when

running PRISM on models of small size (as shown later in Figure 9.5). Above this threshold size, performance of PRISM model checking runs is equivalent to running PRISM on the Java virtual machine.

The design makes extensive use of established software components. This is due to a desire to not "reinvent the wheel". Therefore, the components used are all well established .NET components as the manual development of any component used, to an equivalent quality standard, would have taken more time than the entire PhD project.

The main components used are:

- **AvalonEdit** [231] is a Windows presentation foundation based code-editor employed in the open-source *SharpDevelop* C# code editor. It support a flexible code highlighting a parsing mechanism which is used for showing the XML based representation of Core BPDs and generated PRISM code. It provides parsing, at the syntax level, of these representations and handles reading and writing these file formats.

- **Northwoods GoWPF** is a Windows presentation foundation based implementation of the GoXam user interface controls for implementing diagrams. It provides data structures for and supports the rendering of a wide range of graph-based structures. Of particular note, in terms of the design of SBOAT, is the support for highlighting sub-graphs (groups of nodes), collapsing and expanding trees in-place (used for manageable presentations of fault trees) and automatic layout of graphs. In addition, it provides support for link (edge) and node annotations and drawing new links or reconnecting existing links, with validation.

- **SBOAT Core BPMN to PRISM Compiler** implements the checking of a number of global Core BPMN structural semantics rules, as described in Section 3.3.2. Furthermore, it provides an implementation of the translation from Core BPMN BPDs to PRISM models as described in Section 5.4.

- **SBOAT Analysis Execution Manager** handles the execution of individual instances of the PRISM model checker. It supports setting various properties of the BPMN analysis engine, parsing the output of analysis runs and the caching of generated statespaces in cases where a model has not changed between analysis runs. Furthermore, it manages the parallel execution of multiple instances of the PRISM model checker.

- **.NET Framework** is a well-established software framework developed by Microsoft that runs primarily on Microsoft Windows. It includes a large library. Programs written for the .NET Framework execute in a

software environment (as contrasted to hardware environment), known as the Common Language Runtime, an application virtual machine that provides services such as security, memory management, and exception handling. The class library and the Common Language Runtime together constitute the .NET Framework.

- **PRISM API DLL** is a statically compiled combination of the PRISM engine and Application Programming Interface (API). This component is constructed by means of IKVM.NET [107], a .NET implementation of the Java virtual machine. It should be noted that the specific version of PRISM used for this purpose is modified to allow scheduling generation queries (`multi(...)` keyword) over any number of unbound reward structures.

- **Execution Platforms** can be chosen when performing analysis. SBOAT either supports running analysis locally on the machine hosting the SBOAT tool or it is possible to make use of a cloud compute service. Specifically, SBOAT supports the use of the Microsoft Windows Azure platform which is a cloud computing platform and supporting infrastructure, created by Microsoft, for building, deploying and managing applications and services through a global network of Microsoft-managed datacenters. The Windows Azure Platform provides an API in the form of a client-side managed class library which encapsulates the functions of interacting with the services. This allows for distributing PRISM instances over a large number of processor cores, along with extensive features to manage the workload of each core.

9.2.1 User Interface

The user interface of SBOAT is shown in figure Figure 9.2.

The user interface consists of a tool-bar at the top of the application which is used to load and save models, export statespaces, start a series of analysis runs or obtain limited help. On the left hand the main modelling area is a pallet of Core BPMN elements which can be dragged into a main modelling area to construct models. Sequence flows between Core BPMN elements are constructed by clicking on an element and then clicking on different element to connect the two. If the structural semantic rules do not allow a connection, the cursor icon indicates this and a tool-tip is provided. A group of elements may be selected and then marked as a pool and any valid links that cross a pool boundary automatically become message flows. Right clicking on a link

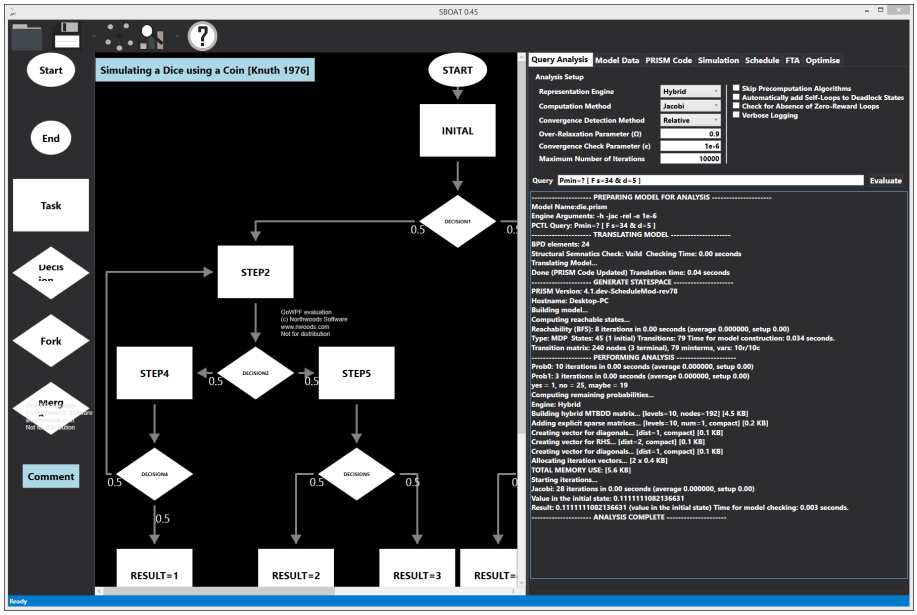


Figure 9.2: *SBOAT version 0.45 Overall User Interface (enlarged)*

or element allows for the annotation of the element. Decision gateways may be annotated with probabilities and tasks and sequence flows may be annotated with reward structures.

On the right hand side of the user interface is a series of 7 tabs. These have the following features:

1. **Query Analysis** tab (Shown) implements the execution of standard Probabilistic Computation Tree Logic (PCTL) queries as demonstrated in Section 5.5. The upper half allows setting PRISM engine options (sensible defaults are chosen), and a text-box allows for input of a PCTL query in the PRISM query language format. Pressing an "evaluate" button starts up a PRISM instance to resolve the query. The results are shown in the box below. In the case when a qualitative query is false a sequence of labelled nodes are returned and these are then matched to the source model (where all nodes and transitions have unique label). The GoWPF based user interface then highlights a path through the model that leads a violation.

2. **Model Data** tab shows the current XML based representation of the model. This is updated in real time as a model is edited and can be exported using a button.

3. **PRISM Code** tab shows the last generated PRISM code representation of the current model. This can be updated by means of a tool-bar button.

4. **Simulation** tab allows for generating traces produced by a random walk through the BPD. At each state it provides information on the accumulated reward values. This mechanism is implemented by means of a random walk algorithm which follows sequence flows through a model. When encountering stochastic gateways, the simulation first randomly chooses a label and then chooses a specific labelled outflow depending on the associated probability. In the case of regions of parallel execution one of the parallel paths are randomly chosen. This tab can be useful for a quick examination of model behaviour.

5. **Schedule** tab implements the schedule generation analysis described in Section 6.1. In the top of the tab there are two boxes, one lists the non-deterministic actions of the model and a button allows moving actions to the other box, indicating that they are to be part of the schedule. For each element in the schedule list of actions a number may be associated, indicating how many times the action should be performed. A number of controls allows for setting PRISM engine options and finally a query text box and output area allow for schedule query specification and output of results. Note that once a schedule has been generated, the highlight mechanism of the GoWPF controls is used to highlight the optimal schedule.

6. **FTA** tab implements fault tree analysis and allows for setting PRISM engine options, has a button to start the analysis and an output area showing the activity of PRISM while the analysis is being performed. Note that using the Fault Tree Analysis (FTA) requires that at least two states have been annotated with fault-continue or fault-stop faults. Once FTA has been completed the output is displayed as a tree by means of the GoWPF controls.

7. **Optimise** tab implements the optimisation analysis. In this tab the usual PRISM engine settings can be input along with mutation rates and population sizes. In addition, a text file must be loaded containing a list of functional PCTL constraints.

In general using SBOAT involves constructing an annotated Core BPMN model, choosing the analysis one wishes to perform and then starting the analysis. It may be desirable to set a number of analysis settings first if one expects the analysis to be particularly complex.

9.2.2 Main data structures

Within SBOAT the modelled BPD is stored as an *adjacency list* [80]. This is implemented using *generics*, allowing a wide range of data to be annotated to either nodes or links. Likewise the structural semantics verification rules associated with individual elements is implemented in a fashion that allows for easy replacement. This makes it possible for future versions of SBOAT to support other modelling languages and therefore this implementation allows nodes and edges to be of any data type, where edges implement a simple interface that defines connections to nodes. This flexibility also supports almost any type of reward annotation to be stored in nodes or edges (although only floating point numbers are used for now). Furthermore, the data structure is mutable and cloneable, making it possible to build a suitable data structure for a wide range of possible graph based process languages. Internally, the data structure keeps a dictionary from nodes to a unordered list of edge elements. Core BPMN are stored as XML and parsing a file involves building the graph data structure as elements are parsed for the XML tree.

The other data structures are mostly trivial, with the main other, non-analysis specific data structure, being the execution log, which records debug information and the output of PRISM instances.

9.3 SBOAT Performance Evaluation

Analysis in SBOAT is done using the PRISM PCTL query language. The central idea is that SBOAT manages the execution of multiple PRISM instances so as to perform efficient analysis. This is focused on parallelising the checking of multiple queries on an individual model (reusing the same statespace) or performing the same query over multiple models. This allows SBOAT to efficiently perform fault tree generation, strategy generation, and, to a lesser degree, business process optimisation.

A limitation of this presentation of the SBOAT tool is that the specific cases on which the tool was employed can not be included in this thesis. Therefore enlarged versions of the examples from the previous chapters have been created by hand to explore the scalability of the analysis. These models are inspired by the type of models encountered in working with the industrial partner.

The performance evaluation shown here, when not otherwise stated, was performed on an Intel Core i7-4810 Processor with 4 cores and supporting 8 concurrent threads of execution and 32GB of RAM.

9.3.1 Bounded Rewards

An unusual feature, in terms of model checking, of this analysis framework is the use of parametrised rewards. The statespace of these models enjoys a large degree of symmetry, as can be seen in Figure 5.3. Performing analysis of enlarged variants of a model highly similar to the one given in Section 3.4, albeit with five bounded reward structures, illustrates, in Figure 9.3, how the analysis of a typical time to completion query scales when employing bounded rewards.

The large degree of symmetry present in the statespaces of the models checked in Figure 9.3, allows for employing the methods of partial order [119] and symmetry [167] reduction implemented in PRISM. This allow for an efficient internal representation of these models, as evidenced by the near logarithmic growth in the amount of memory needed to represent the statespace. Consequently, this allows for efficient model checking of these models and likewise the growth in checking time is also approximated by logarithmic growth.

The key finding is that beyond a certain size limit, in models where bounded rewards produce most of the growth of the statespace, so much symmetry will be present that there is near logarithmic growth in both checking time and memory usage. This behaviour means that including bounded rewards in a model has little impact on the performance of the model checking process.

When exploring the behaviour of a model, it is frequently desirable to try a range of possible reward bounds. This is implemented in SBOAT by choosing an unbound variable u_i for reward structure i's upper bound. In this case u different models will be developed for each reward structure i with a variable upper bound. This behaviour, described in Section 5.3.1, gives rise, in terms of the analysis, to a range of models. While the number of models generated for real world examples can be very large, in this case the checking is easy to parallelize and additional compute services can be employed to perform the associated model checking in parallel, and the same logarithmic growth as seen

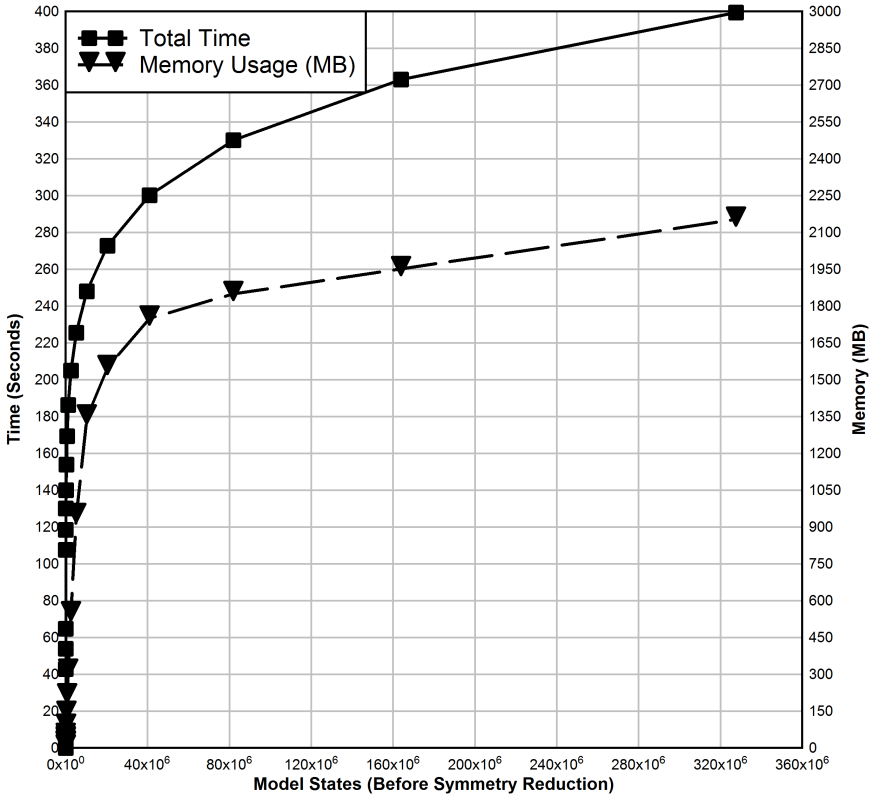


Figure 9.3: *Bounded reward analysis performance of a time to completion query for a scaled up model of the example shown in Section 3.4, making use of five bounded reward structures.*

in Figure 9.3 is observed in checking times of each model. Therefore the effective performance limit of checking a range of models, given sufficient computational resources, is the same as simply checking the model with the largest possible value of the bound.

9.3.2 General PCTL checking

PRISM is a highly capable model checker and able to scale to the analysis of models with sizeable statespaces. Furthermore, the restrictions placed upon Core BPMN models in Section 3.3.2 ensure that they are limited in terms of the parallelism they exhibit. In Figure 9.4, a range of manually constructed models

of various size have been randomly connected by means of message passing to form ever larger models, to determine the scalability of the general PCTL analysis technique.

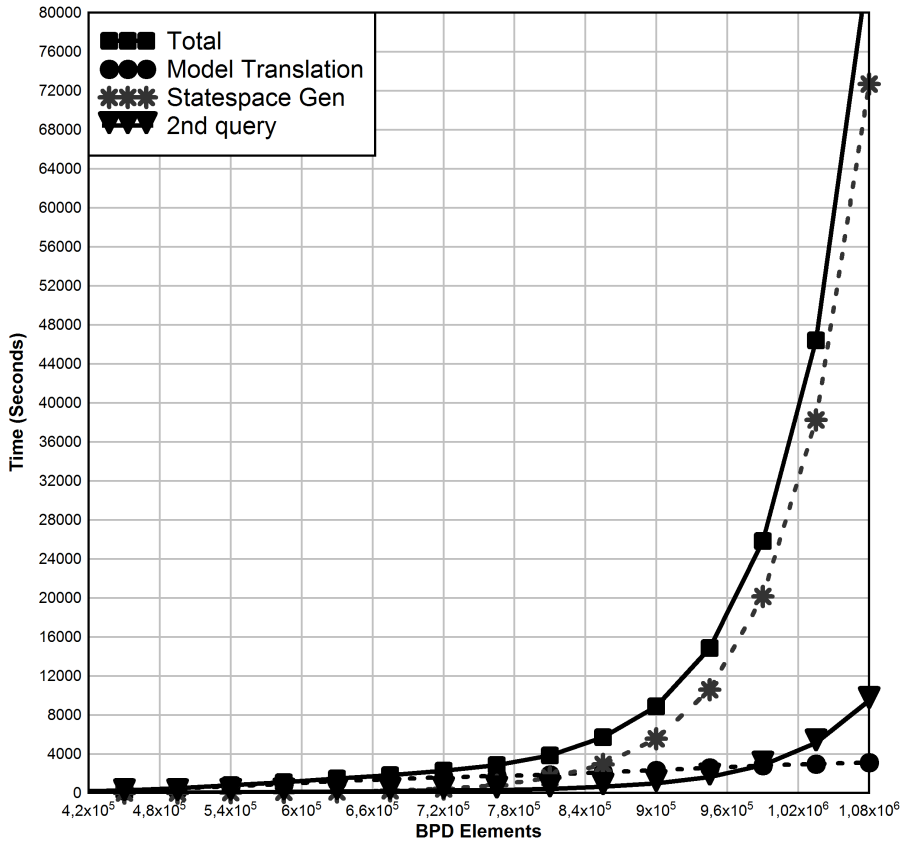


Figure 9.4: *General analysis performance, per step, in terms of execution time for BPD models with an number of elements up to* 10^6.

Note in Figure 9.4, how further (the 2nd query curve) queries only require about 10% of the time taken to perform the initial query. This is due to the fact that once the statespace has been generated, SBOAT ensures it is reused for subsequent queries. A set of further queries to be performed can be executed in parallel such that given sufficient computational resources any number of additional queries only incur an additional 10% overhead.

In line with PCTL model checking theory, the model checking problem is polynomial in the size of the system statespace S. Furthermore, the translation of BPDs to PRISM models while subject to polynomial growth, constitutes only a small fraction of the time taken to perform model checking. The largest model explored in Figure 9.4 has a million BPD elements and exhibits on average 1.5 two-way message exchange between BPDs pools and features at most three parallel regions within each pool. This level of parallelism seem to reflects a scaled up version of the typical business processes encountered by the industrial partner.

An interesting case is the checking of BPDs with a number of elements more in line with the real-world business processes encountered by the industrial partner where the number of elements in a BPD is less than 3500 and which feature a similar degree of parallelism as described above. In this case the checking results are shown in Figure 9.5.

The erratic performance seen in Figure 9.5, for models of small size has the noteworthy feature that the behaviour of checking times and memory usage closely tracks each other and even falls for models with a larger number of states. This is likely due to PRISM finding opportunities for statespace reduction which lead to less memory used and improved checking times. However, another possibility is that the large cache on the platform used is playing a role and leading to this behaviour as it is not seen to the same degree for large models. This result is consistent and highlights how in general model checking of small systems has highly unpredictable performance. Further analysis where a range of cache sizes is employed should make clear which effect is dominant in this analysis.

9.3.3 Scheduling analysis

This analysis, in essence, involves performing a standard PCTL query, but due to the need to keep more of the statespace in memory the limiting factor when performing this analysis is to a greater degree the amount of memory available. This is seen in Figure 9.6 where more data points could not be generated within the memory limit of the platform used.

Figure 9.6 also highlights the close relationship between compacting the statespace and the analysis performance achieved. Larger statespace representations simply take longer to examine. The use of the `multi(...)` keyword, used when generating adversaries which optimise multiple reward structures does not allow for quite as efficient a statespace representation as when performing standard PCTL queries. While this checking should in principle be as efficient as perform-

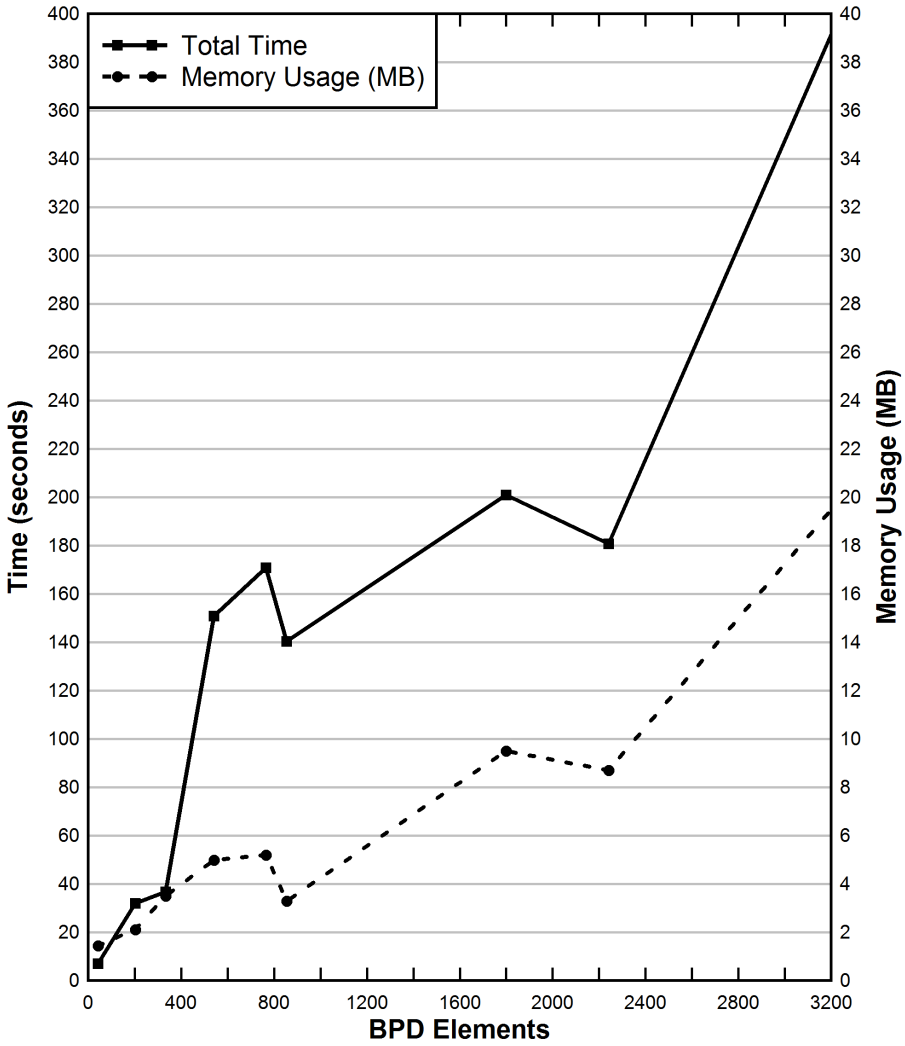


Figure 9.5: *General analysis performance in terms of memory and execution time for models of small size.*

ing standard analysis of PCTL queries, in practice the implementation of this feature in PRISM is still under development and, in particular in more aggressive application of statespace reduction techniques, is not currently possible.

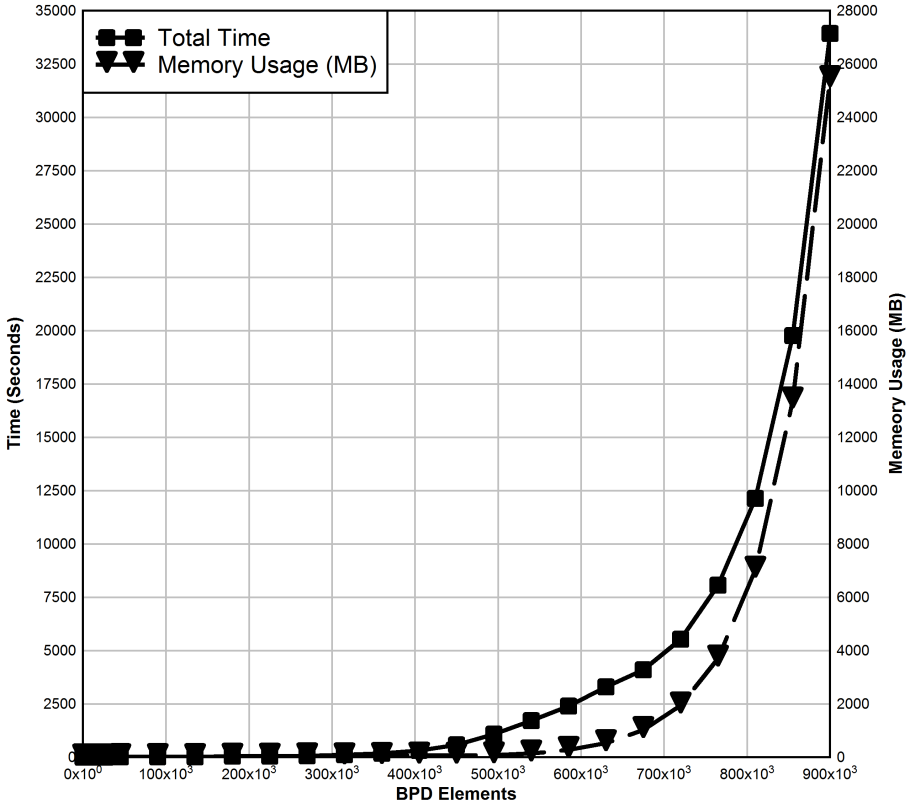


Figure 9.6: *Scheduling analysis performance in terms of memory and execution time for enlarged variants of the robot example given in Figure 6.3, where schedules are generated to optimise two unbound reward structures and an order of* 250 *does of drugs.*

9.3.4 Fault Tree analysis

The FTA inherently involves constructing a fault tree which, whether done manually or in an automated fashion, has a complexity bound of $O(2^n)$. Using the technique for automated FTA described in Section 7.3 involve performing, for each node in the fault tree, a PCTL query for each reward structure of the model and a single lower-bound probability query. However, once the statespace of a model is generated evaluation of each query is relatively inexpensive. Furthermore, each of these queries can be parallellised, meaning that the bulk of the computational demand is the statespace generation itself. This is evident in Figure 9.7.

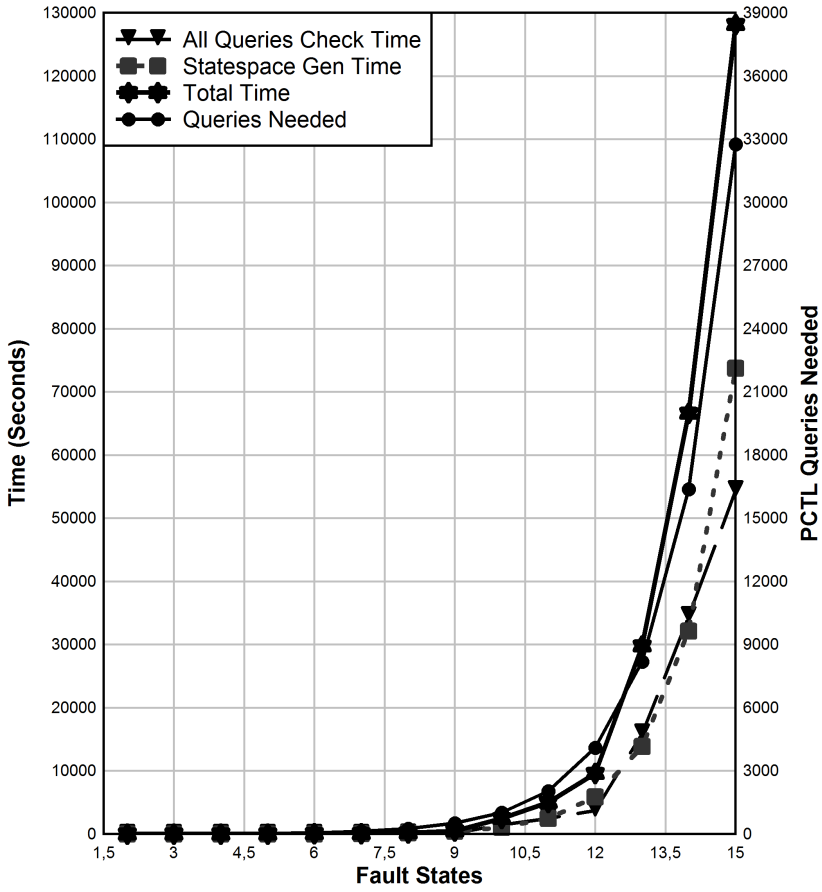


Figure 9.7: *FTA performance in terms of memory and execution time for the doctor example given in Section 3.4.*

In SBOAT when FTA is performed the tree data structure of the GoWPF framework is initialised and starting from the maximal combination of all faults, nodes are added to the tree. The model which corresponds to the top node is model checked and the generated PRISM model is cached and employed for all further model checking runs. Next, for each node in the fault tree the appropriate model checking is performed to determine relevant probabilities and rewards and the result is stored in a hash table with keys based on the alphabetical order of fault terms. When a node is generated which contain a previously identified combination of faults no model checking is performed, instead a hash table lookup is used to obtain the appropriate data. Finally, once construction of the

fault tree is complete the fault tree is rendered using the GoWPF framework and displayed to the user, where areas with identical results are "collapsed" from view.

9.3.5 Optimisation

The performance of each individual PCTL check needed when performing optimisation has already been explored in Section 9.3.2. However, as is common for evolutionary algorithms the time taken to perform each optimisation search is highly unpredictable, the focus in the design of SBOAT is on doing several optimisation searches in parallel. As the individual searches are completed, their results are compared, and if no significant improvement is being discovered, the pathologically slow searches are terminated. A plot of the average time taken to perform analysis at each generation along with the associated average statespace size is shown in Figure 9.8.

It is clear from Figure 9.8 that the analysis times are highly unpredictable and production of this plot required the use of 512 CPU cores on the Azure cloud platform. In addition, the analysis was set up so that when 80% of the cause checking a given population had fallen idle, generation of that population was terminated.

In practical use of the SBOAT tool, once an optimisation run has been completed the final optimal BPD is rendered using the GoWPF framework and displayed to the user.

9.4 Chapter Summary

This chapter introduced SBOAT, the software implementation of the unified framework presented in this thesis. This tool implements the modelling formalism of Core BPMN in the form a GUI modelling tool. SBOAT employs PRISM as the underlying model checking with a focus on employing multiple instances of PRISM in parallel to achieve performance gains in complex analyses.

Experimental results with a series of semi-randomly generated models of various size are presented and discussed.

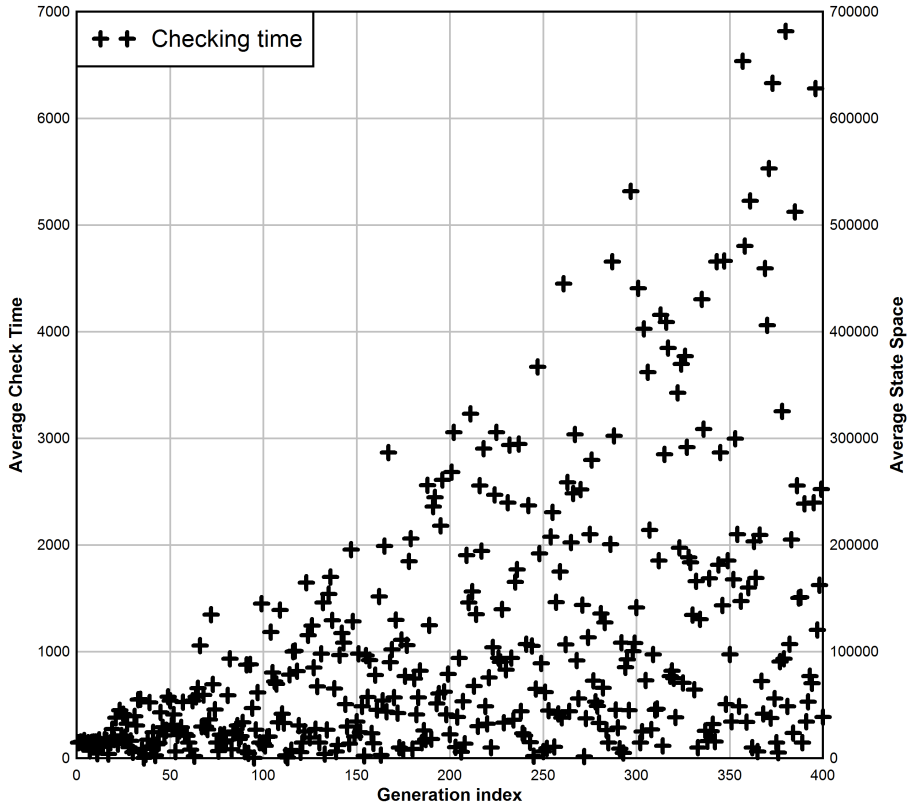


Figure 9.8: *Optimisation Analysis Performance for a standard BPD with 1000 elements and a population size of 512 per generation. This plot is produced over combined analysis runs where* $r_{resequence} \in [0.25, 0.65]$ *and values of* $r_{parallelize} \in [0.1, 0.2]$.

Chapter 10

Conclusions

"The term 'informatics' describes our discipline better than 'computer science', as it is now concerned with communicating and informing as well as with calculation. The most long-lived and famous informatic structure is the von Neumann machine, which gave rise to an impressive series of languages and theories. But the von Neumann machine treats primarily sequential computing on a single machine, and does not scale up to explain modern informatic behaviour."

(Robin Milner 2009)

Overview

This chapter concludes the thesis and provides a brief summary of its main contributions. The approach of using model checking to analyse stochastic business processes is evaluated in terms of the thesis's objectives in modelling, analysis and the various extended analysis. In each case the main strengths of the approach developed is discussed, together with areas that need further refinement. During this discussion some possibilities for further development of this framework are identified. Some final remarks address the broader perspectives of the work presented in this thesis.

10.1 Contributions

This thesis is motivated by the problems described in Section 1.1. While it was not possible to incorporate in this thesis more information about the specific workflows where the industrial partner *Intelligent Hospital Systems* products have been deployed, the overall character of examples presented reflect simplified cases of the industrial problems encountered.

The specific requirement was for a unified framework in which business processes can be specified, verified and analysed in a precise fashion while only requiring limited technical knowledge for an end user. This motivated the decision to develop a unified framework in which a concise stochastic business process modelling language would be tightly coupled with an analysis based on formal methods. The ultimate choice of the formal method of Probabilistic Computation Tree Logic (PCTL) model checking was motivated by consideration that this approach was the one most likely to allow for fully automated analysis. Once the modelling language and associated analysis had been developed, this enabled the construction of a number of advanced analyses which addressed issues such as scheduling and Fault Tree Analysis (FTA). Further, a requirement to suggest improved business processes at design-time - given an initial model of a business process - was explored within the established framework. A software implementation of these ideas was developed which offered a graphical design of business processes and automated the execution of the various analyses.

Specifically, the main contributions of the thesis are:

- An overview of the current state of business process modelling languages with a particular focus on their potential for formal analysis and their ability to capture business processes which exhibit stochastic behaviour. (Chapter 2)

- A formalised sub-set of the Business Process Model and Notation (BPMN) language was developed as a graph-based language. This language was based on a trace semantics corresponding to traversal of the underlying graph structure. This language was then extended with quantitative and stochastic annotations. (Chapter 3)

- A high-level overview of the current formal method approaches which would allow for quantitative and qualitative analysis of stochastic systems was developed with a main focus on the strengths and weaknesses of the chosen method of quantitative probabilistic model checking. Additionally, an overview of the current state of the art probabilistic model checking tools were given and the Probabilistic Symbolic Model Checker (PRISM) model checking tool was selected as the tool which was best suited for the analysis of business processes. (Chapter 4)

- The mechanism for analysis itself was to convert business process models into PRISM models, a process which fundamentally involved the mapping of a graph-based language to a block-based language representing a Markov Decision Process (MDP). This allows the PRISM model checker to employed to analyse models by means of the logic PCTL. (Chapter 5)

- Inherent in the process of MDP model checking is the construction of adversaries that optimise quantitative properties of a MDP. These adversaries can be considered as encoding schedules of non-deterministic actions within a business process. As the PRISM tool performs model checking over all possible adversaries, this allows for its use in a technique which relates adversary generation to scheduling a multiset of actions. Combined with optional qualitative constraints, an optimal schedule can be generated for executing actions within a stochastic business process subject to arbitrary constraints expressed in PCTL.(Chapter 5)

- The analysis framework allows for the development of a method for the construction of fault trees. This is achieved by extending the developed modelling formalism to include the modelling of faults in a business process. The structure of a fault tree containing all possible combinations of these faults is defined and an algorithm presented which determines their probabilities and the quantitative properties of all combined fault states. (Chapter 7)

- By employing parallel execution of the analysis developed in this thesis an approach to design-time optimisation of business processes is constructed. The approach does require the explicit definition of functional constraints required of a future improved business process. Optimisation then takes the form of an evolutionary algorithm which develops candidates for an improved business process with respect to freely defined quantitative properties. (Chapter 8)

- A software implementation of the unified framework, Stochastic BPMN Optimisation Analysis Tool (SBOAT), was developed. This tool implements the modelling formalism of Core BPMN in the form a GUI modelling tool. This tool manages the execution of analyses by means of PRISM, and provides feedback to the user on the result of the analysis. Experimental results with a series of artificial models were performed to determine the performance of an implementation of the framework. (Chapter 9)

Overall, the work of this thesis shows that for an upfront cost of a minor amount of additional information added to business process models such as reward structures or points of possible failure, a substantial amount of information about the behaviour of both the functional and the non-functional quantitative properties of a stochastic business process can be obtained.

10.2 Evaluation of Objectives

This thesis set out to investigate how to specify verify and optimise business processes within a unified framework. Specifically, a set of objectives were defined in Section 1.2 based on the requirements of the industrial partner. In the following subsections each of these is evaluated.

10.2.1 Modelling Objectives

The first set of objectives is focused on developing a modelling formalism that can capture business processes in a fashion which:

1. Allow for the modelling of both non-deterministic and probabilistic decision points. (Objective 1a)

2. Allow for modelling numerical data and resources associated with a business process. (Objective 1b)

3. By means of the imposition of structural semantics serve as a basis for the formalisation of an established business process modelling language which is familiar to business practitioners. (Objective 1c)

As the primary goal of this thesis was to enable formal analysis of business processes it was essential that the modelling formalism used had a formally defined semantics, that was free from ambiguities in its behaviour. Of the three main established business process modelling languages, BPMN and Unified Modelling Language (UML) cannot be said to have a formally defined semantics, preventing their use as an underlying structure to model business processes within a unified analysis framework. Yet Another Workflow Language (YAWL) does have a formally defined semantics, however the semantics of YAWL which are defined by means of a Coloured Petri Net [294] is extremely complex [252] to the extent that it prevents efficient formal analysis. In general the limitations of current business processes languages are elaborated upon in Chapter 2. This motivated the choice to develop a small concise business process modelling language which had a clear semantics close to a well established formal language. Specifically Core BPMN, the developed language, draws inspiration from Communicating Sequential Processes (CSP) to develop a clear semantic base. In addition to allowing for formal analysis a formal semantics provides the basis for an extensible architecture where extensions, in the form of additional constructs, can be precisely defined and their effect on the language understood.

Objective 1a was motivated by the fact that real-world processes inevitably exhibit stochastic behaviour. Fundamentally this stochastic behaviour arises from the *uncertainty principle* which places fundamental limits on the extent to which the future state of any physical system can be determined. While medical workflows are not commonly dependent on quantum mechanical processes, there is a large degree of uncertainty even in the most routine of medical practices. Beyond the medical domain it remains the case that business processes have an underlying stochastic nature either due to either the physical processes on which they depend, or the unpredictable nature of human behaviour.

The overview of different business process modelling approaches presented in Chapter 2 highlights the fact that support for stochastic behaviour in established modelling formalisms is limited. None of the three main established business process modelling formalisms, BPMN, UML or YAWL directly support modelling stochastic systems. This motivated the need for extending a business process modelling language to incorporate stochastic behaviour. MDPs provide a mathematical framework for modelling decision making in situations where

outcomes are partly random and partly under the control of a decision maker. In essence, this allows for capturing two types of unpredictable behaviour, namely that which is probabilistic and that which is non-deterministic in a fashion which allows them to be independent of each other or combined. This allows for an intention preserving stochastic semantics to be imposed on business processes, where choices can be made but their outcomes may not be entirely predictable. The key limitation of MDPs is that they are discrete systems and stochastic or non-deterministic behaviour can only be exhibited in transitions between states of the system. However, this is in line with common business process modelling languages where fundamentally series of discrete tasks are combined into workflows. For this reason it was decided that a modelling formalism that would be well-suited to the type of workflows motivated by the situation described in Section 1.1 should incorporate MDP type behaviour. The development of Core BPMN directly incorporates MDPs as the underlying model of stochastic behaviour within a business workflow and therefore Objective 1a can be considered to be adequately addressed in this thesis.

Objective 1b imposes the requirement to annotate business processes with data and while this is already permissible in established business process modelling languages. These allow for data to be associated with models, however their lack of a formal semantics does not provide a mechanism to determine the change in these data values as the business process is executed. As the intention of this thesis is to provide a unified framework which provides both for the specification and analysis of business processes, data associated with steps in a business process must be given a semantic interpretation. This is necessary to be able to determine the evolution of these data structures as the business process executes. The choice made in this thesis is to employ the domain of real numbers to capture data associated with a business process. This is an extremely expressive structure able to model a wide range of underlying data. However, the approach used in this thesis imposes the requirement that each separate data structure must be monotonic and consequently it should be noted that a key limitation of the current framework is that, while separate data structures are free to either increase or decrease, processes where a single data structure is both incremented and decremented in the course of execution cannot be modelled. In spite of this limitation Objective 1b has still broadly been met as data associated with real-world business processes can frequently be broken down into separate monotonic structures.

Objective 1c has led to the choice to employ BPMN as the inspiration for the business process language developed. This was motivated primarily by studies that identified a compact subset of the language which was overwhelmingly the most commonly used part of the language. Further, the industrial partner had experience with BPMN and there exists a number of guides aimed specifically at employing BPMN in the medical industry [237], [248], [249].

The modelling language presented in this thesis is a reduced subset of the full BPMN language, with the primary aim of formalising this language, and making models described in this notation amenable to formal analysis. This work has been focused on the *core* BPMN subset, which as discussed in Sections 3.3.2 and 3.3.4 can frequently be used to simulate many constructs of the full BPMN language. Many of the omitted constructs, such as specific tasks types which in the full BPMN language exist in a range of forms covering user tasks, manual tasks, service tasks, etc., can be captured using a *type system* to allow for the basic task presented to function as a base type for the specific variants of this element. The wider variety of events in full BPMN which allow for multicast messages and error handling where control flow jumps to one or more pools, can also be addressed by sub-typing the flow relations with appropriate structural semantic rules ensuring meaningful connections between elements. As discussed in Section 3.3.1, many omitted control flow structures can be addressed through pre-processing of BPMN models before analysis.

However, it should be noted that a few constructs involving abstract processes or the inclusive *OR* join are not included in the language developed. While this omission was not found to be a limitation in the medical workflows encountered by the industrial partner, and frequently business processes that make use of this construct use it erroneously [122], there are undoubtedly business processes which cannot be described without this construct.

In general, a wide range of process modelling languages has been developed in recent years and a significant proportion of these make use of directed graphs as their underlying mathematical structure. Therefore, the process of constructing a formalised version of BPMN, by means of the notion of *process graphs* amenable to formal analysis can easily be adapted to other languages which have a fundamentally graph-based structure.

Of particular note are UML statecharts, also known as a UML state machines, these are an enhanced realization of the concept of a finite automaton expressed in UML notation. The description of how a process works (e.g. a business workflow) is organized so that an entity, or each of its sub-entities, is always in one of a number of possible states, and there are well-defined transitions between all states. While not as commonly used to model business processes as BPMN, the key difference between between Core BPMN and UML statecharts is primarily notational and UML statecharts models can be translated to PRISM models in much in the same fashion. By simply replacing the notion of pools and *message passing* with the notion of separate *statecharts* and employing UML's notion of *action synchronization* between these statecharts a formalisation for UML statescharts can be constructed which is similar to what is done here for the core subset of BPMN. Fundamentally UML statecharts can be described by using elements of *process graphs*, or by directly mapping Core BPMN to UML statechart

elements. The structural semantic rules needed to make UML statecharts conform with a reasonable interpretation of the poorly defined UML semantics would be broadly similar to what is done here for BPMN. It should be noted that UML models do allow guards on their transitions and these can be mapped directly to PRISM *action guards* in the same fashion as the direct mapping of guards added to reward exhaustion states. The mechanism for stochastic branching of process graphs (Section 3.2.2) can be used without modification and likewise the reward functions from Definition 3.7 and Definition 3.8 can be easily applied.

Indeed this approach to formalisation of UML is similar to the work by Jansen et. al. [148] which formalises UML statecharts and extends them with probabilities in a fashion similar to that developed here. Their framework also adds data (in the form of rewards) to a UML model, making it possible to analyse the resources consumed by a system by means of the PRISM model checker. The similarity of this approach suggests that the approach taken here to BPMN has at least been pursued by other researchers.

Having discussed the pros and cons of this modelling approach it should be noted that the fundamental limitation of model checking, and of all model-based approaches, is that verification is only as good as the model of the system. Furthermore, the use of modelling languages with synchronisation constructs allows for building complex workflows by modelling sub-components that interact via message passing, allowing the manageable construction of complex business processes by composing components. This allows the Core BPMN language used in this thesis to scale to the description of large systems, and allows these systems to be modelled by separate teams and composed to perform analysis of enterprise-wide behaviour. Further, despite their formalisation it has been possible to ensure that they remain a graphical representation that can be readily employed by business analysts.

It should be noted that while the choice of modelling formalism was originally motivated by problems in the healthcare industry it can easily be employed in other industries, for example in the Baked Goods industry, of which an example is shown in [3].

In conclusion, the modelling approach taken in this thesis has the following strengths and weaknesses:

Main Strengths:

- A concise graphical BPMN based modelling formalism with formal semantics.
- Amenable to model checking via PCTL.

- Captures stochastic and nondeterministic business processes, via an MDP style semantics.
- Able to associate real valued data with nodes and states in a model.
- Extensible to capture other graph based business process languages, via the underlying *process graph* formalism (UML statecharts are particularly well suited).

Main weaknesses:

- The full BPMN language is not supported, in particular inclusive-join gateways are not allowed for.
- Employing MDPs as the underlying modelling limits modelling to discrete systems and stochastic or non-deterministic behaviour can only be exhibited in transitions between states
- Each individual associated data structure is limited to being monotonic

10.2.2 Analysis Objectives

The second set of objectives are focused on developing an analysis technique which allows for:

1. Functional safety properties which assert that the system always stays within some allowed region, such as defined by regulatory requirements, of business processes. (Objective 2a)
2. Non functional qualities of business processes such as timing properties or the determination of resources consumed by business processes. (Objective 2b)
3. Probability bounds on functional and non-functional properties for business processes which exhibit stochastic behaviour. (Objective 2c)

In established engineering disciplines, the reliability and accuracy of predictions of system behaviour are determined by the fidelity of the mathematical models on which they are based. Combined with the extent to which the necessary calculations employing them can be performed without error. The traditional approach to analyse the performance and resource usage properties of business processes by means of simulation do not provide sufficient certainty. In particular when addressing sensitive business processes where safety is a key element, such as those found in the healthcare industry (Section 1.1). Here, strong guarantees must be produced for execution times and resources consumed, and the analysis developed must allow for exact verification often of complex combined properties.

The evaluation of the main techniques for formal analysis, as opposed to simulation, presented in Chapter 4 covers the techniques of theorem proving, static analysis and model checking. A key requirement is for a fully automated approach to formal analysis which excludes theorem proving as a potential technique due to the common need for theorem provers to require manual assistance to complete proofs. The large degree of similarity between static analysis and model checking, as highlighted by the papers [204], [259], makes the choice between these approaches more difficult. The key trade-off is between the size of models that can be tackled and accuracy of the results provided. Static analysis, as it applies to general systems verification, operates over an abstraction of a system and consequently is not always able to produce unambiguous answers and in cases when a verification property is violated it not be possible to produce a counter example. The abstraction of static analysis allows the approach to tackle large statespaces, however it is in the nature of business processes that they are relatively small compared to other domains, such as software verification or systems biology, where static analysis is commonly used. This makes model checking, where an explicit statespace representation is exhaustively explored to perform analysis, a feasible option.

A key advantage of being able to choose model checking is that precise quantitative results can be obtained from the analysis as it explores all states where quantities of interest change. Similarly when considering models with stochastic elements, the complete exploration can provide accurate determination of probability bounds on the occurrence of states of interest, in the cases when bounds exist. This provides excellent results with regard to Objectives 2b and 2c as the results are precise with regard to the Core BPMN models used as input. Likewise in the case of functional properties it can be determined with certainty whether qualitative properties, for example encoding safety constraints, are violated. Furthermore, in the case when qualitative properties are violated, model checking can provide counterexamples in the form of a sequence of tasks and decision gateway choices that lead to a property violation. For these reasons Objective 2a can be considered to be satisfied with regard to accuracy and utility.

By employing model checking for formal verification, properties can be analysed individually during the design process, which allows new properties to be analysed as they are determined to be of importance. This is a key factor when employing these methods in practice, as typically business analysts will not be aware of the full range of properties a business process must satisfy. Instead, this thesis suggests an approach where business processes can be explored throughout their design by means of a fully automated analysis of their properties. This was found be be of great use by the industrial partner because the partner was working in an environment were customer demands and some regulatory requirements were subject to frequent change.

The choice of PCTL, specifically the PCTL* variant, for expression of qualitative or quantitative properties of interest is a rich logic which is able to encode a wide range of properties of interest. There were no properties of interest to the industrial partner that could not be encoded using PCTL, although in the case of resource constrained processes it was necessary to employ the analysis mechanism that allowed for bounded rewards to be included in the analysis. This mechanism produces models with a very large degree of symmetry and which are therefore relatively neutral with regard to the performance of the analysis as PRISM is able to employ symmetry reduction that means that including bounded rewards only produces a modest increase in the effective statespace size. In practice the industrial partner found that this enables the effective provisioning of resources allocated to business processes.

A key weakness in the choice of PCTL is the requirement for business users to employ PCTL queries, implemented in the PRISM query language, to specify properties of interest. Furthermore, business users may misunderstand the verification process as providing absolute guarantees. However, it should be stressed that the method for analysis presented provides firm probability bounds and mean levels of resource usage, but, as in any stochastic system, behaviour far from the mean is possible. However, the nature and significance of these potential flaws in this method are no different from those that attend the use of applied mathematics in other engineering disciplines.

The core analysis of functional and non-functional properties of a business process is able to scale to reasonable (less than 1 day) verification of a Core BPMN Business Process Diagram (BPD) model with a statespace of up to a million elements (Section 9.3.2). This bound includes the time taken to translate a Core BPMNmodel into the PRISM language and generate its statespace and compute an initial PCTL query. Further, queries only require inspection of the statespace, represented as a Multiple Terminal Binary Decision Diagram (MTBDD), which typically requires about 10% of the time taken to determine the initial query. A set of further queries to be performed can be executed in parallel such that given sufficient computational resources any number of additional queries only incur an additional 10% overhead.

The performance of the core analysis holds the potential for further refinement by performing a degree of abstraction of the business process before analysis. Here, once a specific query has been formulated, a reduced and abstracted version of the business process would be generated that only captures states where the properties of interest change. This would reduce the statespace generated by the business process and allow for verification of even larger systems. The likely approach to successfully achieve this would be to develop a Counter Example Guided Abstraction Refinement (CEGAR) approach of which a specific probabilistic version is being developed as an extension for PRISM [114]. While

substantial effort has gone into this work it is still immature and not yet suited for practical use in analysing business processes. However, it should be noted that, in this case, a key drawback is that the generated statespace is unique to the query performed and performing additional queries incur the full overhead of generating a new query specific statespace. It is likely that a hybrid approach employing non-abstracted analysis for models of modest size and switching to abstracted analysis in cases where the analysis time exceeds a user specified bound, would provide the best of both worlds.

In conclusion, the analysis approach taken in this thesis had the following strengths and weaknesses:

Main Strengths:

- Exact analysis and verification results can be obtained, as opposed to simulation results.
- Possible to provide counterexamples, in cases of property violation.
- Properties of interest can be analysed independently
- Separate queries are amenable to parallelisation.
- Generally checking of models is feasible for BPDs with up to one million elements.
- Models can be checked for a range of resource bounds in an efficient manner due to symmetry reduction, which allows for resource provisioning.

Main weaknesses:

- Model checking is fundamentally limited by the statespace explosion problem.
- Properties to be analysed must be expressible in PCTL.
- To employ the framework business users must employ PCTL queries to specify properties of interest.
- Analysis results can be complex for some business users to understand as they express bounds on expected values.

10.2.3 Extended Analysis Objectives

The third set of objectives focuses on meeting a specific set of advanced analysis objectives as dictated by the industrial partner. These include:

1. Automatic synthesis of execution schedules for business processes which are subject to combined functional and non-functional requirements. (Objective 3a)

2. The generation of industry standard fault trees which can assist in the analysis of the effects of combined faults within a business process. (Objective 3b)
3. The optimisation of existing business processes. (Objective 3c)

In general these were addressed by employing the basic qualitative and quantitative analysis technique as a building block in various ways to enable extended analyses which seek to address this set of objectives.

10.2.3.1 Scheduling Analysis

In this thesis the term schedule is defined as the sequence of actions to be taken from an initial state in the system to obtain the optimum of one or more quantitative properties associated with a model. The approach to addressing Objective 3a in this thesis employs the core analysis to perform a query which expresses all possible scheduling possibilities of a chosen set of actions encoded in a disjunction combined PCTL query. These queries can be combined with a freely defined constraint query which can encode properties which must, or must not, hold for a desired schedule.

Fundamentally, being able to determine the quantitative properties of a business process allows for the determination of the effect of specific sequences of actions on the performance of the business process by means of optimal adversaries. During the development of this thesis a new version of the PRISM model checker (version 4.1) was released which greatly simplified the process of adversary generation in the case of generating adversaries which optimise multiple reward structures associated with a model. This means that generating a schedule which optimises multiple rewards is now done in a single model checking run, specifically PRISM performs model checking over all possible adversaries and selects the optimal choice. These changes in PRISM allow for very efficient schedule generation as the performance of adversary generation is no different then simply performing a normal model checking query using the core analysis framework, as the process of adversary generation is inherent to the model checking of MDPs.

For a business practitioner to determine a schedule it simply requires specifying a multiset of non-deterministic actions which can be scheduled, along with a PCTL query encoding any constraints on the desired schedule. The automatic translation of this multiset and the constraints simply involves building a PCTL query that is linear in the size of the set of *actions* and *constraints* of the scheduling problem. Hence this adds no significant performance penalty when performing this analysis.

With regard to building a unified framework the determination of schedules has been successful. It is a fully automated analysis that requires minimal input from a business user to construct and the labelled induced Discrete-time Markov Chain (DTMC) which form the schedule can readily be mapped back to the original Core BPMN process to produce a clear indication of how the optimal schedule will be executed in terms of the modelled business process. Combined the positive results in this area suggest that Objective 3a has been met.

It should be noted that the limitation in PRISM 4.1 that adversaries can only be generated over a maximum of two unbound PRISM reward or probability queries is only present to allow the generation of Pareto Curves modifying the freely available PRISM source code to remove this restriction, as done in SBOAT, allows for schedule generation with regard to any number of unbound reward or probability queries.

It should be noted that this procedure can neither deal with cases where only part of the actions are controllable while others are uncontrollable disturbances, which would require strategy construction in 2 1/2 player games, nor partially observable processes, as the state-dependent adversary assumes perfect identifiability of the current process state. Both restrictions place some limitations on applying this approach to practical implemented processes.

In conclusion, the scheduling analysis approach taken in this thesis had the following strengths and weaknesses:

Main Strengths:

- Schedules can be defined for quantitative properties, where qualitative constraints are also observed.
- Schedule generation can be done efficiently as a single model checking operation for multiple goals and constraints.

Main weaknesses:

- Schedules are limited to those for which goals and constraints can be expressed in PCTL.
- A technical limitation in PRISM 4.1 is that adversaries can only be generated over a maximum of two unbound PRISM reward or probability queries, without forking the underlying PRISM engine.
- No support for limited information or not modelled disturbances.

10.2.3.2 Fault Tree Generation

The generation of fault trees is not typically part of Business Process Management (BPM) approaches. In the case of this thesis it was a requirement of our industrial partner. However, it was discovered that this type of system analysis with a view to suggesting system failures that can arise from individual system faults can provide valuable input when considering safety aspects of a business process. This approach is focused on design time modelling of a business process before it has been implemented. The motivating context described in Section 1.1 of medical workflows are subject to regulation which in many cases explicitly require FTA to have been performed before a business process is put into practice. Automated FTA can save time and cost by ensuring that business processes, for which the construction of fault trees is a regulatory requirement, can have fault trees generated for them throughout their development. When a design change produces a fault tree which implies that the design is unlikely to gain regulatory approval, this design change can be excluded at that point in development.

In a fashion similar to the approach taken in schedule generation, the approach taken to FTA is to employ the core analysis framework to construct PCTL queries describing possible combinations of faults. Faults have been incorporated into the Core BPMN language so as to be a strict addition to the basic Core BPMN modelling constructs, allowing faults to be added and removed with only local modifications to a task. In terms of the SBOAT implementation of this analysis faults can be added through simple annotations to tasks. Once an initial fault tree has been generated where all faults are combined by *OR* gates, designers can modify the gates to better reflect realistic interpretations of how the faults may combine. The central strength of this stochastic model checking approach is that the probabilities of combined faults or failures occurring will reflect not only the combined probabilities of individual faults but will also account for the base stochastic nature of the underlying business process, providing useful guidance on the likely bounds of the probability of failure.

The incorporation of quantitative information into the fault tree provides further useful information for a business process designer in that it can suggest mean values of the expected time to failure, its associated cost and in general the values of any reward structures of interest at the point of failure. This information can be useful in informing testing of business processes as it can provide guidance on when faults are expected during testing or the magnitude of their impact.

While the approach presented successfully meets Objective 3b in that it produces industry-standard fault trees and even is able to annotate them with quantitative data, a fundamental problem is the need to generate the underlying fault tree skeleton which involves generating generating all possible subsets of the sets of

faults which has a complexity which is exponential in the number of fault states. Further, while the statespace for the faulty model must only be generated once PCTL queries must be generated corresponding to each of the sub-sets.

While these individual queries can be parallellised, the growth in the computational burden of this approach means that large amounts of computing resources must be devoted to the task. This is the motivation behind the ability to inject fault states for specific tasks and not simply generate a maximum fault tree where every task has a potential to be faulty. Despite this there are clear limitations to the scalability of the approach. In spite of this the technique was found to be useful by the industrial partner as the same complexity bounds apply to manual construction of a fault tree. In the SBOAT implementation the assistance given in fault tree construction is still of utility compared to a strictly manual approach to fault tree generation. Hence, Objective 3b can be considered to be met when compared with the industrial partner's current manual alternative.

In conclusion, the fault tree generation approach taken in this thesis had the following strengths and weaknesses:

Main Strengths:

- Adding faults to tasks is a straight-forward local annotation to tasks in Core BPMN models.
- Quantitative data about the system's state can be determined at points of failure.

Main weaknesses:

- The scalability of this approach is fundamentally limited by the exponential growth in the size of the fault tree as more fault states are added.
- Fault trees involve an inherent creative element in determining whether a combination of faults actually lead to a failure and what the nature of the failure is.

10.2.3.3 Optimisation

Objective 3c has been addressed through the use of an evolutionary algorithm approach. This objective is a highly ambitious form of Business Process Re-Engineering (BPR) which seeks to omit the human element in redesigning business processes and instead move the burden of such work to an algorithmic approach. At design time there is only a limited amount of information available

to suggest what form an optimal business process might take. The specific choice of an evolutionary algorithm is motivated by the enormous statespace of possible business processes that can be devised to meet a specific business objective.

The practical application of this approach requires extensive specification of what is required of an optimal business process. Frequently requiring specifications at a level of detail that goes beyond what is commonly used in the business community. An algorithmic approach requires the definition of explicit information which would be obvious to anyone taking part in the design or implementation of a business process. This limitation is central to attempting to generate optimised business processes at design time. However, when working with our industrial partner, it became clear that the process of making a full specification for a business process explicit could be very beneficial. While it could at times become more verbose than would be typical for such specifications, frequently it was found that some elements of the specification was ambiguous or allowed for improvements which, while typically minor, were nevertheless non-obvious.

From a theoretical perspective it is possible to achieve optimisation of business processes through the iterative modification and model checking of business processes. The model checking step, which is fundamentally the same as in the core analysis, is computational quite feasible for business processes of a realistic size. However, the vast number of variant child processes generated when employing an evolutionary algorithm which must each be checked both for structural semantic conformance and individually model checked involving the generation of a unique statespace in each case, does set limits on its practical utility. It is the contention of this thesis that an evolutionary algorithm approach provides the most efficient search of the space of possible business process' variants in the case where the goals for optimisation can be defined as both quantitative and qualitative properties.

The key weakness in the specific approach adopted here is in the generation of variants. The approach taken closely follows the basic principles of evolutionary algorithms and future work in this area should be devoted to finding improved techniques for variant generation. Variants should ideally be well-formed by construction such that if one starts from a well-formed business process it is at least highly unlikely that derived variants will violate the structural semantic rules defined.

The performance of this approach, while relatively easy to parallelize, is in its current form computationally expensive. A future development in this area would be to improve the generation of variants done as part of the optimisation process. Notions of the bi-similarity of business processes developed by Kunze et. al. [165] could be employed to broaden the variants generated while simultaneously increasing the likelihood that a variant will be structurally sound. The ability

of this definition to preserve the required properties of a system while allowing enough freedom to explore a wide range of possible changes would substantially enhance this method. This effectively means allowing variants which are bi-similar to be considered, as opposed to those simply generated by mutation and crossover. This would require some refinement of these early notions of bi-similarity, but should allow the optimisation approach to scale to much larger systems and for the discovery of optimisations of greater novelty.

Within the bounds of the requirements of Objective 3c this thesis has managed to develop a strategy for producing improved business processes at design time without manual intervention. The traditional BPM approach is cyclic and requires a process to be implemented before optimisation of a process is considered. In this case considerably more information is available and an evolutionary algorithm approach, as demonstrated by Medeiros [184] where data from an instrumented implemented process is used to derive a model, is highly successful at finding optimised models. However, the work in this thesis highlights that without this information the amount of optimisation which can be performed is much more limited.

In conclusion, the optimisation approach taken in this thesis had the following strengths and weaknesses:

Main Strengths:

- The approach successfully finds improved processes within the specification given, at design time.
- Defining the needed goals is for an improved process can be beneficial in itself.
- The approach is amenable to parallelisation and sufficient computational resources allow it to be applied to models of up to one million BPD elements per generation per day, with a population size limited only by the amount of parallel computation resources available.

Main weaknesses:

- A detailed explicit specification of the constraints upon an improved solution must be defined.
- Variants generation must be finely tuned to quickly converge on improved processes.
- Computationally expensive

10.3 Wider Perspectives

The BPM field employs informatics concepts to support the design, enactment, management, and analysis of business processes. Although the practical adoption of BPM concepts has been widespread [276], it is a field where a clear definition of terms are missing and in many areas it lacks a formal mathematical foundation.

There are many different kind of ways to describe a business process. By developing Core BPMN this work is placed within the business aspect of process modelling, which is consistent with this project seeking to full research objectives from an industrial partner. Beyond BPMN and the other feasible of modelling language option of UML statecharts, the approach developed here of formal analysis of business processes could be transferred to other domains where the same basic ideas of this thesis could be employed. For example, biological systems, where languages broadly similar to BPMN such as the such the biological signalling pathway modelling language *Kappa* [85] could be enhanced with a model checking based approach to where in particular the FTA analysis developed would be of considerable utility. Alternatively, the telecom focused *SDL* modelling language [145] could be a domain where the basic ideas of the thesis and in particular accurate performance analysis and scheduling analysis approach could be of great benefit. In general, the core concepts presented here sketch a path for the general conversion of graph-based modelling languages into a block-based structure which is amenable to quantitative probabilistic model checking in cases when stochastic behaviour, inherent in real world processes, is present.

While formal analysis of business processes is a growing field in the literature, however most approaches to not include stochastic behaviour and only a limited number address general quantitative properties. This thesis seeks to show that for a number of different applications a stochastic model checking approach can be successfully applied. The use of model checking and PCTL places the work in the context of one of central problems of informatics namely addressing concurrency through an effective mathematical theory. These methods have seen extensive development, and the underlying model checking approach is now able to scale up to handle the analysis of relatively simple systems such as business processes. A comparison can be made between the development of business processes and the general development of software systems. However, as business processes are systems of lesser complexity, it becomes possible to explore what can be achieved with model checking beyond verification of basic properties. In this way, this thesis explores the future of what can be achieved in model checking of software systems. The advanced analyses, and in particular the optimisation method presented, potentially allow for a substantial improvement in the development of business processes and hopefully could inform the optimisation, or synthesis, of software systems. In particular the cross-fertilisation of ideas from static analysis

and model checking, as illustrated by [204] and [259], continue to enhance the scalability and accuracy of each other allowing ever bigger systems to be tackled by this approach.

In a business context, with regard to strategy generation and optimisation, the methods presented here provide an appealing route to reducing the burden of performing change management. Change management programs typically involve putting an improved business process into practice. Crucially, the methods presented here make change management a process of simply defining the goals for the improved process, and then automatically deriving the process that will best meet these needs and, through strategy generation, determining who are to perform which actions in the new process. Crucially, this derivation is an automated step and a set of constraints could be envisioned which would encode that an actor should experience a certain level of disruption in their workflows.

Finally, this work in the area of business process optimisation, despite its limitations, provides a mathematical foundation to suggest that Frank Gilbreth's [113] nearly one-hundred year old vision of *finding the one best way to do work* is flawed in its suggestion that there exists a single optimal business process for an enterprise in a given industry. The optimisation methods presented here suggest that looking for ideal business processes is akin to searching for an ideal organism in a given biological ecosystem. Instead, there will, in any sufficiently complex market, be multiple niches which specific business processes are able to exploit. This work demonstrates a general method for the determination of high performance safe business processes which dominate in their given niche.

In conclusion, the overall objective of the thesis, namely, developing an unified framework for the specification, verification and optimisation of business processes has been broadly successful.

10.4 Directions for future work

There are a number of areas of this framework that have the potential for further development.

Future developments of modelling aspect framework could make use of PRISM's mechanism to support reward annotations expressed as algebraic expressions of data structures associated with the model. The reason this is not included in this framework is that no such demand was uncovered while working with the industrial partner, and in all explored cases, rewards could be expressed as

simply positive real numbers. While the capability for complex rewards could be readily introduced into this framework, they would come at a cost of a significant increase in the complexity of the statespaces to be analysed. However, the limiting monotonicity requirement is inherent to the model checking approach and will remain a fundamental limitation of this framework.

With regard to building models from sampled data, the approach presented here is informal, based on observing the difference encountered upon refining bins. This approach could be enhanced to employ interval probabilities and interval MDPs, thus being able to conservatively quantify the worst-case estimation error. This would be a relatively simple addition, and have a limited effect on the analysis complexity.

Additional further work in terms of modelling could focus on the replication of business processes in the style of the μ-calculus [164], this would allow for modelling business processes with an element of mobility. This is potentially possible within this framework through PRISM's support for module renaming, which allows duplication of modules. This is done at a textual level, so any identifiers, including action labels, constants and functions used in the module definition can be changed during conversion to PRISM models. While no need was identified in the cases of the industrial partner for this capability, this would allow sub-processes in a business process to make copies of themselves and be relocated to other points in a business process. If this capability was added the main drawback would be a significant increase in the complexity of the statespaces used within the verification process and a consequent growth in the time taken to perform model checking.

In terms of the analysis aspect, the notion of temporal logics is not a traditional business management concept, and an area for future research would be the construction of a logic grounded in business concepts for the expression of desired properties. Within this framework this logic would be employed by translation into the PRISM property specification language. To make this logic readily usable in a business setting an extension of the BPMN-Q a visual query language [36] to include probability and reward queries would likely be an ideal approach. Alternatively the Business Property Specification Language (BPSL) of Xu et. al [293] allows for a similar visual logic specification approach.

In terms of the scheduling analysis aspect future work could focus on the experimental additions to the PRISM model checker [70] which provide a mechanism, by means of turn-based stochastic multi-player games, to resolve adversaries for multiple actors in a system. This would provide the underlying framework for building separate *strategies* (a more appropriate term than schedules in the case of game theory) for competitive systems where separate sub-systems are working towards different goals. When these methods are more mature they would be

relatively straightforward to incorporate into this work. This would allow for the generation of *strategies* in a competitive business environment. Here separate enterprises would each be modelled individually together with a additional model describing the market in which they are competing. For each enterprise a set of actions they are able to perform is defined and the stochastic game can then be analysed using model checking to determine a *strategy*. PCTL queries would encode one enterprise seeking to optimise properties of interest, while competing enterprises are seeking to minimise the same quantities for the original enterprise. The ultimate goal would be to develop a strategy that would allow one enterprise to outperform another with regard to properties of interest. It should be noted the work of [70] also allows for modelling imperfect information although in its present form it is not readily applicable to the work of this thesis.

In terms of the fault tree analysis aspect further work could focus on handling faults in message passing between processes as these poses a similar set of problems to those explored in this thesis. Lost messages and situations where messages are delayed or arrive out of sequence could also be accounted for. A future research direction in this area is to develop an approach to these faults using a preprocessing step which models queues of lost messages which have associated probabilities of being sent each time a transition is made in the system. Further messages can be received by other recipients than intended, leading to further complexity. Combined with advanced rewards, this allows for modelling and deriving fault trees for considerably more complex business processes. Handling faults in message exchange is also crucial in dealing with large scale enterprise-wide faults where a lack of successful communication is frequently a key factor in failure [263].

An approach by Mukherjee et. al. [195] to generating fault trees from maintenance logs can be greatly enhanced by the work presented here. Specifically, the methods of the framework allow business processes models with a considerably greater degree of freedom in their behaviour to be built from logs prior to fault tree generation. This creates fault trees which better reflect the system and which include data (in the form of rewards). This could be a direction for future research.

In terms of the optimisation aspect the performance of this approach while relatively easy to parallelize is in its current form computationally expensive. A future development in this area would be to improve the generation of variants done as part of the optimisation process. Notions of the bi-similarity of business processes developed by Kunze et. al. [165] could be employed to broaden the variants generated while simultaneously increasing the likelihood that a variant will be structurally sound. The ability of this definition to preserve the required properties of a system while allowing enough freedom to explore a wide range of possible changes would substantially enhance this method. This effectively means

allowing variants which are bi-similar to be considered, as opposed to those simply generated by mutation and crossover. This would require some refinement of these early notions of bi-similarity, but should allow the optimisation approach to scale to much larger systems and for the discovery of optimisations of greater novelty.

With regard to defining appropriate requirements for the optimisation of a business process, a problem exists in that the initial specifications are often too weak (e.g., they may be vacuously satisfied), or too strong (e.g., they may allow too many system behaviours). Work has been done in this area by Chatterjee et. al. [69] on the synthesis of an environment in which a system will satisfy a given specification. Specifically, they show that an ω-regular specification ϕ can be realized if, and only if, there exists a winning strategy in a certain parity game constructed from ϕ. If ϕ is not realizable, they then construct an environment assumption ψ such that $\psi \rightarrow \phi$ is realizable. The focus of this work is on the automated refining of a specification and not on optimisation of the system itself. Future work in this area could focus on incorporating their approach to allow for some degree of automated refinement of the functional and non-functional constraints supplied by a user. This should allow for better exclusion of variants which have little hope of evolving into viable optimisations.

Therefore, future research in the area of optimisation area should acknowledge that optimising business processes requires at a minimum that a test version of a business process be implemented and instrumented for analysis so that future improved variants can be determined. In this case, a combination of the workflow mining techniques of Medeiros [184] with the design time approach developed in this thesis, is likely to help iterative process improvement more rapidly converge on an ideal maximally improved process. This would extend the notion of workflow mining to not just determining the nature of an existing process but also allow automated improvements, minimising the manual burden of the BPR life-cycle.

Future work in the area of the framework presented in this thesis in general, could enable a more automated approach to the formalisation of a graph-based language and thereby extend the automated nature of this method to also encompass the addition of new languages, through a meta language which describes the source language and allows for the specification of associated structural semantic constraints.

Appendix A

Quantitative Model Checking Theory

This Appendix presents a formal description of quantitative model checking drawn primarily from the *Principles of Model Checking* by Christel Baier and Jost-Pieter Katoen [39], and *Advances in Probabilistic Model Checking* by Marta Kwiatkowska and David Parker [171].

A.1 Discrete-Time Markov Chains

Fundamentally, probabilistic model checking operates by performing model checking on a Discrete-time Markov Chain (DTMC). This is a simple probabilistic system model, which models systems whose behaviour at each point in time can be described by a discrete probabilistic choice over several possible outcomes. A DTMC consists of discrete states representing specific configurations of a system, and has transitions governed by (discrete) probability distributions over the target states. Each transition is assumed to take a discrete time-step and there are no deadlocks, and all terminating states are modelled with a self-loop.

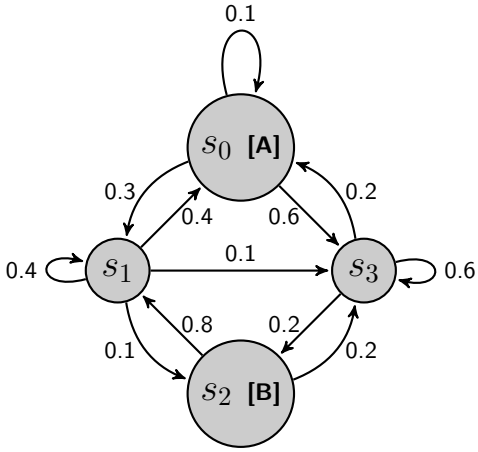
Definition A.1 (Discrete-time Markov chain)
A discrete-time Markov chain (DTMC) is a tuple $\mathcal{D} = (S, \bar{s}, \mathbf{P}, L)$ *where* S *is a (countable) set of states,* $\bar{s} \in S$ *is an initial state,* $\mathbf{P} : S \times S \to [0, 1]$ *is a transition probability matrix such that, for all* $s \in S$*:*

$$\sum_{\forall s' \in S} \mathbf{P}(s, s') = 1$$

and $L : S \to 2^{AP}$ *is a labelling function mapping each state to a set of atomic propositions taken from a set* AP*.*

A DTMC can be represented as a *labelled transition system* (LTS) in which each transition is annotated with a probability value indicating the likelihood of its occurrence. An example of a simple DTMC where $S = s_0, s_1, s_2, s_3$ is shown in fig. A.1(a). Labels $A, B \in AP$ have been associated with states s_0 and s_2 respectively. The transition probability matrix for this example is shown in fig. A.1(b), where each row has a sum of 1 as required by the restriction on transition probability Matrices.

A DTMC captures the evolution of a system by letting each element $\mathbf{P}(s, s')$ of the matrix $\mathbf{P}$ define the probability of a transition from s to s' taking place. In a DTMC, a transition is assumed to take a discrete time-step and means that there is no notion of real time present in the model. Reasoning about any discrete



(a) LTS Representation

$$\mathbf{P} = \begin{pmatrix} 0.1 & 0.3 & 0.0 & 0.6 \\ 0.4 & 0.4 & 0.1 & 0.1 \\ 0.0 & 0.8 & 0.0 & 0.2 \\ 0.2 & 0.0 & 0.2 & 0.6 \end{pmatrix}$$

(b) Transition Probability Matrix $\mathbf{P}$

Figure A.1: *An example of a 4 state DTMC (a) and the corresponding matrix* $\mathbf{P}$ *which defines the probabilities of state transitions (b).*

time structure is possible, by assigning time intervals to states and accounting for the transition steps taken. There are no deadlocks, and all terminating states are modelled with a self-loop.

A DTMC model can be *unfolded* (unwound) into a set of paths. Where a *path* through a DTMC is a non-empty finite or infinite sequence of states $\pi = s_n, s_{n+1}, s_{n+2}, \ldots$ with $\mathbf{P}(s_i, s_{i+1}) > 0$ for all $i \geq 0$. In the example from fig. A.1, if $\bar{s} = s_0$ is chosen, the path sets shown in fig. A.2 are obtained, within which a specific path could be $\pi = s_0 s_0 s_3 s_0$.

The probability matrix $\mathbf{P}$ induces a probability space on the set of infinite paths $Path_s$, which start in state s using a cylinder construction as follows [153]. An observation of a finite path determines a basic event (cylinder). Let $s = s_0$, for $\pi = s_0, s_1, \ldots s_n$, then a probability measure Pr_s^{fin} for the π-cylinder can be defined as:

$$Pr_s^{fin} = \begin{cases} 1 & \text{if } \pi \text{ contains a single state} \\ \mathbf{P}(s_0, s_1) \cdot \mathbf{P}(s_1, s_2) \cdot \ldots \cdot \mathbf{P}(s_{n-1}, s_n) & \text{otherwise} \end{cases} \tag{A.1}$$

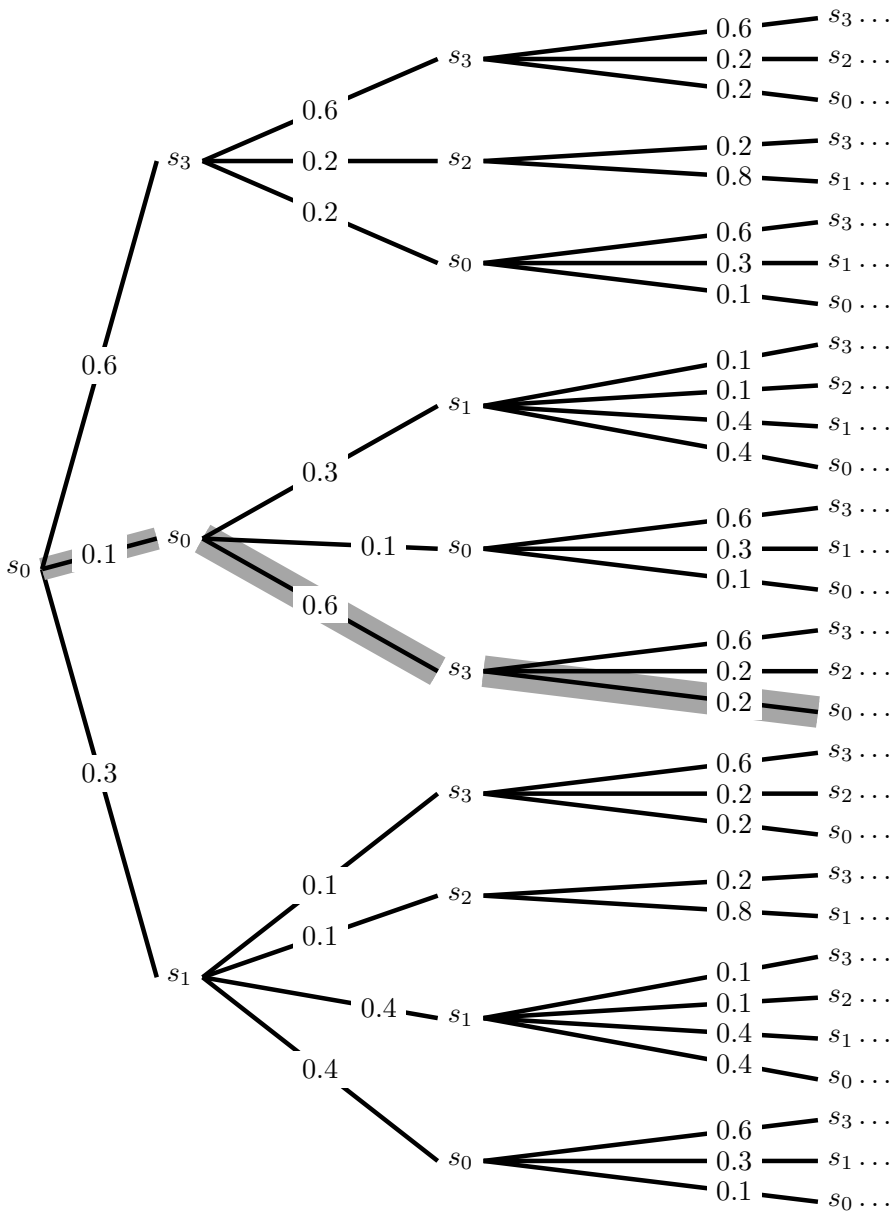


Figure A.2: *3 levels of unfolding of the DTMC from Figure A.1 starting from the starting state $\bar{s} = s_0$. A specific path $\pi = s_0, s_0, s_3, s_0$ is highlighted.*

It is proven by Kemeny et. al. [153] that eq. (A.1) extends to a unique measure Pr_s on the set of infinite paths $Path_s$.

A.2 Probabilistic Computation Tree Logic

Specifications for DTMC models can be expressed in a range of temporal logics. In this description, a probabilistic extension of the temporal logic Computation Tree Logic (CTL) [75] known as Probabilistic Computation Tree Logic (PCTL) [131] is employed:

Definition A.2 (PCTL Syntax)
The syntax of PCTL is as follows:

$$\begin{aligned} \phi &:= \texttt{true} \mid a \mid \phi \wedge \phi \mid \phi \vee \phi \mid \neg\phi \mid \mathtt{P}_{\bowtie p}[\psi] \\ \psi &:= \mathtt{X}\,\phi \mid \phi\,\mathtt{U}\,\phi \end{aligned} \tag{A.2}$$

where a is an atomic proposition, $\bowtie \in \{\leq, <, >, \geq\}$ and $p \in [0, 1]$.

PCTL formulas are interpreted over the states of a DTMC. PCTL replaces the CTL existential and universal quantification over paths with the probabilistic operator $\mathsf{P}_{\bowtie p}[\cdot]$ where $p \in [0, 1]$ is a chosen *probability bound*. Note that in definition A.2 path formulas ψ can occur only within the scope of the probabilistic operator. Formally, the meaning of the $\mathsf{P}_{\bowtie p}[\cdot]$ operator is:

Definition A.3 (DTMC Model checking P Operator)
Given a statespace S, a path formula ψ, and a chosen state $s \in S$ then

$$s \models \mathsf{P}_{\bowtie p}[\psi] \Leftrightarrow Pr_s(\psi) \bowtie p \quad \text{where} \quad Pr_s(\psi) \bowtie p \stackrel{def}{=} Pr_s\{\omega \in Path_s | \omega \models \psi\} \tag{A.3}$$

Where $\bowtie \in \{\leq, <, >, \geq\}$ and $p \in [0, 1]$.

Here, the standard CTL path formulas are included in PCTL:

- *Next state*: $(\mathtt{X}\,\phi)$ which is true for a path $\omega \in Path_s$ if ϕ is satisfied in the next state.
- *Unbounded until*: $(\phi_1\,\mathtt{U}\,\phi_2)$ which is true for a path $\omega \in Path_s$ if ϕ_2 is satisfied at some state along the path and ϕ_1 is true up until that point.

From these, further derived formulae are possible, such as:

- *Eventually*: $\mathtt{F}\,\phi \Leftrightarrow \mathtt{true}\,\mathtt{U}\,\phi$ which can be used to determine *reachability*, i.e. the probability of reaching a state satisfying ϕ.
- *Always*: $\mathtt{G}\,\phi \Leftrightarrow \neg(\mathtt{true}\,\mathtt{U}\,\neg\phi)$ which can be used to determine *invariance*, i.e. the probability of ϕ always remaining true.

A.3 Model Checking DTMCs with PCTL

Qualitative model checking is performed by calculating probability measure $Pr_s(\psi) \bowtie p$ compared to a chosen probability bound $p \in [0, 1]$, which yields respectively true or false. Vardi showed in 1985 [277], that the probability measure $Pr_s(\psi) \bowtie p$ of the set ϕ-paths is indeed measurable.

Quantitative model checking involves calculating the probability bound for the outermost probabilistic operator in a PCTL formula. Hanson and Johnson [131] defined the original PCTL model checking algorithm, as an extension of the original CTL model checking algorithm [75]. The PCTL model checking algorithm takes as input a labelled DTMC $\mathcal{D}$ and a PCTL formula ϕ and proceeds, by bottom-up traversal of the parse tree for ϕ recursively computing the set $Sat(\phi') = \{s \in S | s \models \phi'\}$ of states satisfying each sub-formula ϕ'. For the operators of PCTL to establish if a given state s satisfies ϕ, it is necessary to check $s \in Sat(\phi)$, which the algorithm computes as:

$$\begin{array}{lcl} Sat(\mathtt{true}) & = & S \\ Sat(a) & = & \{s \in S | a \in L(s)\} \\ Sat(\neg\phi) & = & S \backslash Sat(\phi) \\ Sat(\phi_1 \wedge \phi_2) & = & Sat(\phi_1) \cap Sat(\phi_2) \\ Sat(\mathsf{P}_{\bowtie p}[\psi]) & = & \{s \in S | Pr_s(\psi) \bowtie p\} \end{array} \tag{A.4}$$

When computing these sets, it is convenient to view the DTMC as the matrix $\mathbf{P}$ and $Sat(\phi)$ as a column vector $\underline{\phi} : S \to 0, 1$ given by $\underline{\phi}(s) = 1$ if $s \models \phi$ and 0 otherwise. In the case of the next operator $\mathtt{X}\,\phi$ the probabilities for all states can then be computed by a single matrix-by-vector multiplication, written in

vector notation as $\underline{Pr}(\mathtt{X}\,\phi) = \mathbf{P} \cdot \underline{\phi}$. For the until path operator $\phi_1 \,\mathtt{U}\, \phi_2$ the probabilities of $Pr_s(\phi_1 \,\mathtt{U}\, \phi_2)$ are obtained as the unique solution of the *system of linear equations* in the variables $\{x_s | s \in S\}$ as:

$$x_s = \begin{cases} 0 & \text{if } s \in S^{no} \\ 1 & \text{if } s \in S^{yes} \\ \sum\limits_{\forall s' \in S} \mathbf{P}(s, s') \cdot x_{s'} & \text{if } s \in S^{?} \end{cases} \tag{A.5}$$

Where $S^{no} \stackrel{def}{=} Sat(\mathsf{P}_{\leq 0}[\phi_1 \,\mathtt{U}\, \phi_2])$ and $S^{yes} \stackrel{def}{=} Sat(\mathsf{P}_{\geq 1}[\phi_1 \,\mathtt{U}\, \phi_2])$ denote the sets of all states that satisfy $\phi_1 \,\mathtt{U}\, \phi_2$ with probability exactly 0 and 1, respectively, and $S^{no} \stackrel{def}{=} S \backslash (S^{no} \cup S^{yes})$.

The sets S^{no} and S^{yes} are precomputed using conventional fixed point computations. Since the values for the precomputed states are known (0 or 1), the solution of the resulting linear equation system in $|S^{?}|$ variables can be obtained by any direct method (e.g. Gaussian elimination) or iterative method (e.g. Jacobi, Gauss-Seidel). For qualitative PCTL properties, it suffices to use these pre-computation algorithms alone. Note that the pre-computation algorithms determine the exact probability in case it is 0 or 1, thus avoiding the problem of round-off errors that are typical for numerical computation. The time complexity for PCTL model checking over DTMCs is linear in the size of the formula $|\phi|$(number of logical connectives and temporal operators) and polynomial in the size of the state space $|S|$ [131].

A.4 Model Checking Rewards

Let $\mathcal{D} = (S, \bar{s}, \mathbf{P}, L)$ be a DTMC. A reward structure is a pair (r_s, r_t) of functions: a *state reward* function $r_s : S \to \mathbb{R}_{\geq 0}$ mapping a state to the reward acquired per time-step, and a transition reward function $r_s : S \times S \to \mathbb{R}_{\geq 0}$, mapping each transition to the reward acquired as the transition is taken. The rewards can be interpreted in two ways: instantaneous (omitted for simplicity) or cumulated over system execution.

Checking these can be done by extending the logic PCTL [168] to allow for the reward properties by the addition of the *reward operator* $\mathsf{R}_{\bowtie r}[\cdot]$. State formulas $\mathsf{R}_{\bowtie r}[\mathtt{C}^{\leq k}]$ denoting cumulative rewards and $\mathsf{R}_{\bowtie r}[\mathtt{F}\phi]$ denoting reachability rewards, are defined for a state s:

Definition A.4 (DTMC Model checking R Operator)
Given a statespace S, ϕ a PCTL formula, and a chosen state $s \in S$ then

$$\begin{array}{lll} s \models \mathsf{R}_{\triangleright\triangleleft r}[\mathtt{C}^{\leq k}] & \Leftrightarrow & \mathbb{E}_s(X_{\mathtt{C}^{\leq k}}) \bowtie r \\ s \models \mathsf{R}_{\triangleright\triangleleft r}[\mathtt{F}\phi] & \Leftrightarrow & \mathbb{E}_s(X_{\mathtt{F}\phi}) \bowtie r \end{array} \tag{A.6}$$

Where $\bowtie \in \{\leq, <, >, \geq\}$, $k \in \mathbb{N}$, $r \in \mathbb{R}_{\geq 0}$, and $\mathbb{E}_s$ denotes the expectation with respect to the probability measure Pr_s.

The random variables $X_{\mathtt{C}^{\leq k}}, (X_{\mathtt{F}\phi} : Path_S \to \mathbb{R}_{\geq 0}$ corresponding to the two forms of the reward operator are defined, for any path $\omega = s_0 s_1 s_2 \cdots \in Path$ as:

$$X_{\mathtt{C}^{\leq k}}(\omega) \stackrel{def}{=} \begin{cases} 0 & \text{if } k = 0 \\ \sum_{i=0}^{k-1} r_s(s_i) + r_t(s_i, s_{i+1}) & \text{otherwise} \end{cases}$$

$$X_{\mathtt{F}\phi}(\omega) \stackrel{def}{=} \begin{cases} 0 & \text{if } s_0 \models \phi \\ \infty & \text{if } \forall i \in N. s_i \not\models \phi \\ \sum_{i=0}^{min(j|s_j \models \phi)-1} r_s(s_i) + r_t(s_i, s_{i+1}) & \text{otherwise} \end{cases} \tag{A.7}$$

Model checking of the reward operator is similar to computing probabilities for the probabilistic operator, and follows through the solution of recursive equations for $\mathsf{R}_{\triangleright\triangleleft r}[\mathtt{C}^{\leq k}]$ or a system of linear equations for $\mathsf{R}_{\triangleright\triangleleft r}[\mathtt{F}\phi]$. Model checking rewards has the same time complexity as checking probabilistic properties.

A.5 Markov Decision Processes

Markov Decision Processes (MDPs) generalise discrete-time Markov chains with the addition of non-determinism. Non-determinism is used to model unknown environments, where such distributions are not known. It is also used to model concurrency, where it represents the different possible interleavings of multiple components operating in parallel. Formally, a Markov decision process is defined as:

Definition A.5 (Markov Decision Process)
An MDP is a tuple $MDP = (S, \bar{s}, Act, Steps, L)$ where S is a set of states, $\bar{s} \in S$ is an initial state, Act is an alphabet of actions, $Steps : S \times Act \to Dist(S)$ is a partial probabilistic transition function and and $L : S \to 2^{AP}$ is a labelling function mapping each state to a set of atomic propositions taken from a set AP.

In an MDP, several actions may be available in a given state s, each corresponding to a probability distribution. This set is denoted by $A(s) = \{a \in Act | Steps(s,a) is defined\}$. Like for DTMCs, deadlocks are disallowed and it is assumed that $A(s)$ is non-empty for all $s \in S$. The behaviour of an MDP MDP is as follows. Firstly, a choice between one or more actions from the alphabet Act is made non-deterministically. Secondly, for the chosen action a, a successor state s' is chosen randomly, according to the probability distribution $Steps(s,a)$, i.e. the probability that a transition to s' occurs is $Steps(s,a)(s')$.

Figure A.3 shows an example MDP $MDP = (S, \bar{s}, Act, Steps, L)$ with states $S = \{s_0, s_1, s_2, s_4\}$, an initial state s_0 and alphabet of actions $Act = \{a, b, c\}$ and labels $L = \{INIT, HEADS, TAILS\}$. In this example, state s_1 has a non-deterministic choice between two actions, b and c.

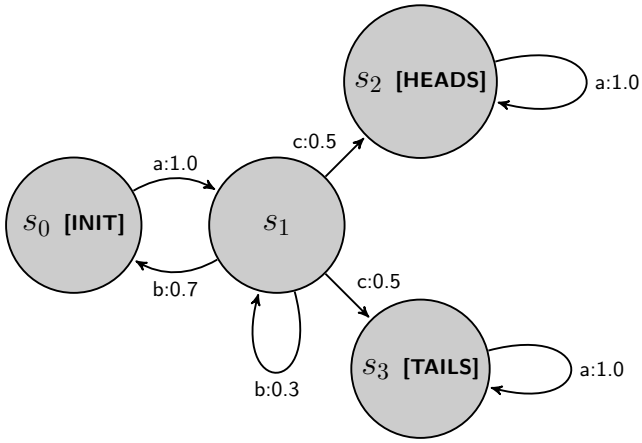


Figure A.3: *An LTS representation of an 4 state example MDP.*

An infinite path through an MDP is a sequence $\omega = s_0 a_0 s_1 a_1 \ldots$ where $s_i \in S$,$a_i \in A(S_i)$ and $Steps(s_i, a_i)(s_{i+1}) > 0$ for all $i \in \mathbb{N}$. A finite path $\pi = s_0 a_0 s_1 a_1 \ldots s_n$ is a prefix of an infinite path ending in a state. $Path_s$ and $Path_s^{fin}$ denotes the sets of all infinite and finite paths from state s respectively, and $Path$ and $Path^{fin}$ denotes the corresponding sets of paths from any state. For a finite path π the last state of π is denoted $last(\pi)$.

A.6 Adversaries over MDPs

Formal reasoning about MDPs requires a probability space over infinite paths. However, a probability space can only be constructed once all the non-determinism has been resolved. Each possible resolution of non-determinism is represented by an adversary (also known as strategies), which is responsible for choosing an action in each state of the MDP, based on the history of its execution so far.

Definition A.6 (Adversary)
An adversary of an MDP $MDP = (S, \bar{s}, Act, Steps, L)$ *is a function* $\sigma : Path^{fin} \to Dist(Act)$ *such that* $\sigma(\pi)(a) > 0$ *only if* $a \in Act(last(\pi))$. *An adversary* σ *is memoryless if* $\sigma(\pi)$ *depends only on* $last(\pi)$ *and deterministic if the distribution* $\sigma(\pi)$ *always selects a single action with probability* 1.

The set of all adversaries of an MDP MDP is denoted Adv. Under a particular adversary $\sigma \in Adv$, the behaviour of MDP is is fully probabilistic and can be captured by an *induced DTMC*, denoted MDP^σ, each state of which is a finite path of MDP.

Definition A.7 (Induced DTMC)
For an MDP $MDP = (S, \bar{s}, Act, Steps, L)$ *and adversary* σ, *the induced DTMC is* $MDP^\sigma = (Path^{fin}, \bar{s}, \mathbf{P}, L')$ *where*

- *for any* $\pi, \pi' \in Path^{fin}$*:*

$$\mathbf{P}(\pi, \pi') = \begin{cases} \sigma(\pi)(a) \cdot Steps(last(\pi), a)(s) & \text{if } \pi' = \pi a s, a \in A(last(\pi)) \\ 0 & \text{if otherwise} \end{cases} \tag{A.8}$$

- $L'(\pi) = L(last(\pi))$ *for all* $\pi \in Path^{fin}$

There is a one-to-one mapping between the infinite paths of the DTMC MDP^σ and the infinite paths of the MDP MDP when under the control of adversary σ. This means that the DTMC yields, for a start state s, a probability space, denoted Pr_s^σ over these infinite paths. The induced DTMC MDP^σ has a (countably) infinite number of states. However, in the case of memoryless adversaries, its state space is isomorphic to S and MDP^σ can be reduced to an $|S|$-state DTMC.

Consider the example MDP MDP from fig. A.3 and the (deterministic, but non-memoryless) adversary σ, which picks action b the first time that state s_1 is reached, and then action c the second time. The induced DTMC MDP^σ is shown in fig. A.4.

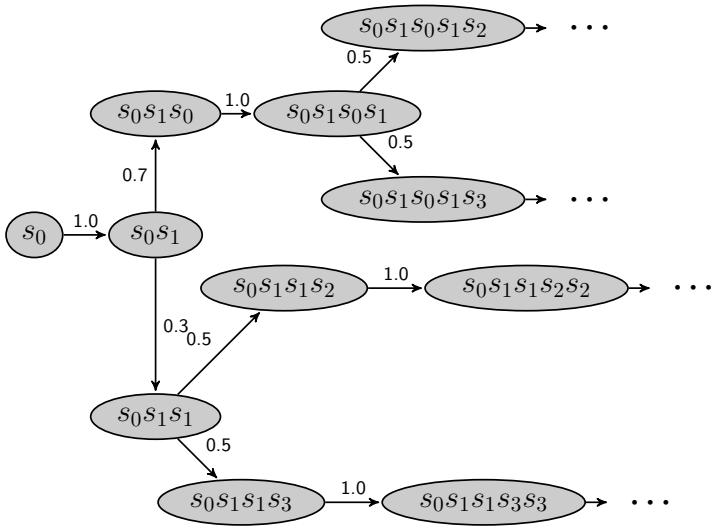


Figure A.4: *The induced DTMC for an adversary of the MDP in fig. A.3.*

A.7 PCTL Model Checking over MDPs

Probabilistic statements about MDPs typically involve quantification over adversaries, so as to establish that some specified event is observed *for all* possible adversaries. The logic PCTL, for example, is defined for MDPs as for DTMCs [46], the key difference being that the semantics of the probabilistic operator contains explicit universal quantification:

Definition A.8 (MDP Model checking P Operator)
Given a statespace S, a path formula ψ, and a chosen state $s \in S$ then

$$s \models \mathsf{P}_{\bowtie p}[\psi] \;\Leftrightarrow\; Pr_s^{\sigma}\{\omega \in Path_s | \omega \models \psi\} \bowtie p \quad \text{for all } \sigma \in Adv \tag{A.9}$$

Where $\bowtie \in \{\leq, <, >, \geq\}$ and $p \in [0, 1]$.

The algorithm for PCTL model checking proceeds as for DTMCs, except for the probabilistic operator. For $\mathsf{P}_{\rhd p}[\psi]$ where $\rhd \in \{\geq, >\}$, this reduces to the calculation of the *minimum probability* $Pr_s^{min}(\psi)$. The case of $\mathsf{P}_{\lhd p}[\psi]$ where $\lhd \in \{\leq, <\}$ is dual, *maximum probability* $Pr_s^{max}(\psi)$:

$$\begin{array}{rcl} Sat(\mathsf{P}_{\rhd p}[\psi]) & = & \{s \in S | Pr_s^{min}(\psi) \rhd p\} \\ Sat(\mathsf{P}_{\lhd p}[\psi]) & = & \{s \in S | Pr_s^{max}(\psi) \lhd p\} \end{array} \tag{A.10}$$

Where $Pr_s^{min}(\psi) = inf_{\sigma \in Adv}\{Pr_s^{\sigma}(\psi)\}$ and $Pr_s^{max}(\psi) = sup_{\sigma \in Adv}\{Pr_s^{\sigma}(\psi)\}$

To describe the computation of these values, attention is restricted to the case of minimum probabilities (maximum probabilities are analogous). If $\psi = \mathtt{X}\,\phi$, we have:

$$Pr_s^{min}(\mathtt{X}\,\phi) = min_{a \in A(s)} \left[\sum_{s' \in Sat(\phi)} Steps(s,a)(s') \right] \tag{A.11}$$

For $\psi = \phi\,\mathtt{U}\,\phi$, the minimum probabilities are the unique solution of:

$$Pr_s^{min}(\psi) = \begin{cases} 0 & \text{if } s \in S^{no} \\ 1 & \text{if } s \in S^{yes} \\ min_{a \in A(s)} \left[\sum_{s' \in S} Steps(s,a)(s') \cdot Pr_{s'}^{min}(\psi) \right] & \text{if } s \in S^{?} \end{cases} \tag{A.12}$$

with S^{no} and S^{yes} denoting the sets of states where the minimum probability is respectively 0 and 1, precomputed in a similar fashion to the DTMC case via a fix-point. The computation of these probabilities can be performed in several different ways; including *linear programming*, *value iteration*, or *policy iteration*.

The logic PCTL can be extended, as for DTMCs, with the reward operator $\mathsf{R}_{\bowtie r}[\cdot]$ in quantitative forms $\mathsf{R}_{min}[\cdot]$ and $\mathsf{R}_{max}[\cdot]$. The difference is that the minimum (or maximum) reward values are computed as expectations.

The time complexity for PCTL model checking over an MDP is (again) linear in the size of the formula $|\phi|$ and polynomial in the size of the state space $|S|$. Like for DTMCs, MDPs can also be verified against Linear Temporal Logic (LTL) properties via the construction of the product with a deterministic *Rabin automaton* for the LTL formula. Model checking reduces to the computation of, maximum reachability probabilities of, a set of end components of the product MDP. The overall complexity for LTL is doubly exponential in $|\phi|$ and polynomial in $|S|$; unlike for DTMCs, this cannot be reduced to a single exponential.

Associated Publications

The following publications where produced as part of this thesis:

[1] Z. N. L. Hansen, P. Jacobsen, L. Herbert, S. J. Pedersen, and S. Frosch, "Challenges in production changeover. example from the baked goods industry", in *Journal of Mathematics, Statistics and Operations Research (JMSOR) secretariat*, (In review), 2014 (cited on page 14).

[2] L. Herbert, Z. N. L. Hansen, and P. Jacobsen, "Automated evolutionary restructuring of workflows to minimise errors via stochastic model checking", in *Proceedings of the Probabilistic Safety Assessment & Management conference*, (Forthcoming 2014 PSAM Conference), 2014 (cited on page 154).

[3] L. Herbert, Z. N. L. Hansen, and P. Jacobsen, "Optimal sceduling of stochastic production complex processes through stochastic model checking: an example from the baked goods industry", in *Notes on Modelling and Management of Engineering Processes*, series Springer book series, (Forthcoming 2014), Springer, 2014 (cited on pages 142, 202, 226).

[4] L. Herbert, Z. N. L. Hansen, and P. Jacobsen, "SBAT: a stochastic BPMN analysis tool", in *Proceedings of the ASME 2014 12th Biennial Conference on Engineering Systems Design and Analysis (ESDA2014)*, (Forthcoming 2014 ESDA Conference), 2014 (cited on page 202).

[5] L. Herbert, Z. N. L. Hansen, and P. Jacobsen, "SBOAT: a stochastic BPMN analysis and optimisation tool", in *Proceedings of the International Conference on Engineering and Applied Sciences Optimization (OPT-i)*, (Forthcoming 2014 OPT-i Conference), 2014 (cited on page 202).

[6] L. Herbert, Z. N. L. Hansen, and P. Jacobsen, "Using quantitative stochastic model checking tool to increase safety and improve efficiency in production processes", in *Proceedings of the 2014 European Safety and Reliability Association ESREL conference*, (Forthcoming 2014 ESREL Conference), 2014 (cited on page 154).

[7] L. Herbert, Z. N. L. Hansen, P. Jacobsen, and P. Cunha, "Evolutionary optimisation of production materials workflow processes", in *Proceedings of the 8th International Conference on Digital Enterprise Technology (DET 2014), Stuttgart*, 2014 (cited on page 176).

[8] L. Herbert, Z. N. L. Hansen, R. Sharp, and P. Jacobsen, "Optimal sceduling of complex processes through stochastic model checking: an example from the baked goods industry", in *Proceedings of the 3rd International Conference on Modelling and Management of Engineering Processes (MMEP 2013)*, Oct. 2013 (cited on page 142).

[9] L. Herbert, K. R. M. Leino, and J. Quaresma, "Using Dafny, an automatic program verifier", in *Tools for Practical Software Verification*, series Lecture Notes in Computer Science, B. Meyer and M. Nordio, Eds., volume 7682, Berlin, Heidelberg: Springer-Verlag, 2012, pages 156–181, ISBN: 978-3-642-35745-9. DOI: 10.1007/978-3-642-35746-6_6 (cited on pages 78, 295).

[10] L. Herbert and R. Sharp, "Towards quantitative evaluation of stochastic pharmacy workflows", in *Proceedings of the 23rd Nordic Workshop Programming Theory*, 2011 (cited on page 14).

[11] L. Herbert and R. Sharp, "Quantitative analysis of probabilistic BPMN workflows", in *ASME 2012 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (IDETC/CIE2012))*, series ASME Conference Proceedings, Jul. 2012, pages 509–518. DOI: 10.1115/DETC2012-70653 (cited on pages 14, 36, 118).

[12] L. Herbert and R. Sharp, "Using stochastic model checking to provision complex business services", in *High-Assurance Systems Engineering (HASE), 2012 IEEE 14th International Symposium on*, Oct. 2012, pages 98–105. DOI: 10.1109/HASE.2012.29 (cited on pages 36, 118, 142).

[13] L. Herbert and R. Sharp, "Optimisation of BPMN business models via model checking", in *ASME 2013 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (IDETC/CIE2013)*, series ASME Conference Proceedings, Aug. 2013. DOI: 10.1115/DETC2013-13047 (cited on page 176).

[14] L. Herbert and R. Sharp, "Precise quantitative analysis of probabilistic business process model and notation workflows", *Journal of Computing and Information Science in Engineering*, volume 13, pages 2–11, 1 Mar. 2013. DOI: 10.1115/1.4023362 (cited on pages 14, 36, 118).

[15] L. Herbert and R. Sharp, "Workflow fault tree generation through model checking", in *Proceedings of the 2013 European Safety and Reliability Association ESREL conference*, 2013 (cited on page 154).

[16] L. Herbert and R. Sharp, "Model-checking business processes", in *Advances in Computational Sciences and Information in Engineering*, series ACIER book series, (Accepted for publication), ASME Press, 2014 (cited on page 142).

General Bibliography

Works cited:

[17] W. M. P. van der Aalst, "The application of petri nets to workflow management", *Journal of Circuits, Systems and Computers*, volume 8, number 1, pages 21–66, 1998. DOI: 10.1142/S0218126698000043 (cited on pages 56, 59).

[18] W. M. P. van der Aalst, "Verification of workflow task structures: a petri net based approach", *Information Systems*, volume 25, number 1, pages 43–69, Mar. 2000, ISSN: 0306-4379. DOI: 10.1016/S0306-4379(00)00008-9 (cited on pages 56, 119).

[19] W. M. P. van der Aalst, "Business process management: a comprehensive survey", *ISRN Software Engineering*, 2013. DOI: 10.1155/2013/507984 (cited on pages 18–20).

[20] W. M. P. van der Aalst, L. Aldred, M. Dumas, and A. H. M. Hofstede, "Design and implementation of the yawl system", in *Advanced Information Systems Engineering*, series Lecture Notes in Computer Science, A. Persson and J. Stirna, Eds., volume 3084, Springer Berlin Heidelberg, 2004, pages 142–159, ISBN: 978-3-540-22151-7. DOI: 10.1007/978-3-540-25975-6_12 (cited on page 29).

[21] W. M. P. van der Aalst, B. F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A. J. M. M. Weijters, "Workflow mining: a survey of issues and approaches", *Data & Knowledge Engineering*, volume 47, number 2, pages 237–267, Nov. 2003, ISSN: 0169-023X. DOI: 10.1016/S0169-023X(03)00066-1 (cited on page 182).

[22] W. M. P. van der Aalst and K. M. van Hee, "Business process redesign: a petri-net-based approach", *Computers in Industry*, volume 29, number 1-2, pages 15–26, Jul. 1996, ISSN: 0166-3615. DOI: 10.1016/0166-3615(95)00051-8 (cited on page 182).

[23] W. M. P. van der Aalst and A. H. M. ter Hofstede, "YAWL: yet another workflow language (revised version)", *Wireless Networks*, volume 30, number 4, pages 245–275, Jun. 2005, ISSN: 0306-4379. DOI: 10.1016/j.is.2004.02.002 (cited on pages 29, 30, 44, 300).

[24] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros, "Workflow patterns", *Distributed and Parallel Databases*, volume 14, number 1, pages 5–51, Jul. 2003, ISSN: 0926-8782. DOI: 10.1023/A:1022883727209 (cited on pages 29, 30, 32).

[25] W. M. P. van der Aalst, M. H. Schonenberg, and M. Song, "Time prediction based on process mining", *Information Systems*, volume 36, number 2, pages 450–475, Apr. 2011, ISSN: 0306-4379. DOI: 10.1016/j.is.2010.09.001 (cited on page 121).

[26] W. M. P. van der Aalst, T. Weijters, and L. Maruster, "Workflow mining: discovering process models from event logs", *IEEE Transactions on Knowledge and Data Engineering*, volume 16, number 9, pages 1128–1142, Sep. 2004, ISSN: 1041-4347. DOI: 10.1109/TKDE.2004.47 (cited on page 182).

[27] W. M. P. van der Aalst, J. Desel, and E. Kindler, "On the semantics of epcs: a vicious circle", in *EPK*, M. Nüttgens and F. J. Rump, Eds., GI-Arbeitskreis Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten, 2002, pages 71–79. [Online]. Available: http://dblp.uni-trier.de/db/conf/epk/epk2002.html#AalstDK02 (cited on page 64).

[28] W. Aalst, "Verification of workflow nets", in *Application and Theory of Petri Nets 1997*, series Lecture Notes in Computer Science, P. Azèma and G. Balbo, Eds., volume 1248, Springer Berlin Heidelberg, 1997, pages 407–426, ISBN: 978-3-540-63139-2. DOI: 10.1007/3-540-63139-9_48 (cited on page 121).

[29] AIS group, Eindhoven University of Technology. (Dec. 2013). CPN tools 4.0, [Online]. Available: http://cpntools.org/ (cited on pages 33, 122).

[30] R. Alur and T. A. Henzinger, "Reactive modules", *Formal Methods in System Design*, volume 15, number 1, pages 7–48, 1999, ISSN: 0925-9856. DOI: 10.1023/A:1008739929481 (cited on pages 101, 105, 107).

[31] American National Standards Institute and Institute of Electrical and Electronics Engineers, "Software engineering standards: glossary of software engineering terminology", New York, NY, USA, ANSI/IEEE 729-1983, 1984 (cited on page 156).

[32] T. Au and T. Stelson, *Introduction to systems engineering and deterministic models*, series Addison-Wesley series in civil engineering. Boston, MA, USA: Addison-Wesley, 1969 (cited on page 21).

[33] A. Awad, G. Decker, and M. Weske, "Efficient compliance checking using BPMN-Q and temporal logic", in *Proceedings of the 6th International Conference on Business Process Management*, series BPM '08, Berlin, Heidelberg: Springer-Verlag, 2008, pages 326–341, ISBN: 978-3-540-85757-0. DOI: 10.1007/978-3-540-85758-7_24 (cited on page 120).

[34] A. Awad, R. Goré, J. Thomson, and M. Weidlich, "An iterative approach for business process template synthesis from compliance rules", in *CAiSE 2011*, series Lecture Notes in Computer Science, H. Mouratidis and C. Rolland, Eds., volume 6741, Berlin, Heidelberg: Springer-Verlag, 2011, pages 406–421, ISBN: 978-3-642-21639-8. DOI: 10.1007/978-3-642-21640-4_31 (cited on page 180).

[35] A. Awad and S. Sakr, "On efficient processing of BPMN-Q queries", *Computers in Industry*, volume 63, number 9, pages 867–881, Dec. 2012, ISSN: 0166-3615. DOI: 10.1016/j.compind.2012.06.002 (cited on page 120).

[36] A. Awad, M. Weidlich, and M. Weske, "Visually specifying compliance rules and explaining their violations for business processes", *Journal of Visual Languages and Computing*, volume 22, number 1, pages 30–55, Feb. 2011, ISSN: 1045-926X. DOI: 10.1016/j.jvlc.2010.11.002 (cited on pages 120, 239).

[37] J. Backus, "History of programming languages I", in, R. L. Wexelblat, Ed., New York, NY, USA: ACM, 1981, ch. The History of Fortran I, II, and III, pages 25–74, ISBN: 0-12-745040-8. DOI: 10.1145/800025.1198345 (cited on page 86).

[38] J. Baeten, "The total order assumption", English, in *Proceedings of the First North American Process Algebra Workshop, Stony Brook, New York, USA, 28 August 1992*, series Workshops in Computing, S. Purushothaman and A. Zwarico, Eds., Springer London, 1993, pages 231–240, ISBN: 978-3-540-19822-2. DOI: 10.1007/978-1-4471-3217-2_14 (cited on page 111).

[39] C. Baier and J.-P. Katoen, *Principles of Model Checking*. Cambridge MA, USA: The MIT Press, 2008, ISBN: 9780262026499 (cited on pages 112, 113, 199, 244, 293, 294, 296–298).

[40] T. Ball, B. Cook, V. Levin, and S. Rajamani, "SLAM and static driver verifier: technology transfer of formal methods inside microsoft", in *Integrated Formal Methods*, series Lecture Notes in Computer Science, E. Boiten, J. Derrick, and G. Smith, Eds., volume 2999, Springer Berlin Heidelberg, 2004, pages 1–20, ISBN: 978-3-540-21377-2. DOI: 10.1007/978-3-540-24756-2_1 (cited on page 88).

[41] R. Banach and M. Bozzano, "The mechanical generation of fault trees for reactive systems via retrenchment ii: clocked and feedback circuits", *Formal Aspects of Computing*, pages 1–49, 2011, ISSN: 0934-5043. DOI: 10.1007/s00165-011-0203-6 (cited on pages 159, 161).

[42] D. Barker-Plummer, J. Barwise, and J. Etchemendy, *Language, Proof, and Logic: Second Edition*, 2nd. Center for the Study of Language and Information/SRI, 2011, ISBN: 9781575866321 (cited on pages 79–81).

[43] G. Behrmann, A. David, K. Larsen, J. Hakansson, P. Petterson, W. Yi, and M. Hendriks, "UPPAAL 4.0", in *Third International Conference on Quantitative Evaluation of Systems QEST*, 2006, pages 125–126. DOI: 10.1109/QEST.2006.59 (cited on pages 91, 143, 299).

[44] G. Behrmann, A. Fehnker, T. Hune, K. Larsen, P. Pettersson, J. Romijn, and F. Vaandrager, "Minimum-cost reachability for priced time automata", English, in *Hybrid Systems: Computation and Control*, series Lecture Notes in Computer Science, M. Benedetto and A. Sangiovanni-Vincentelli, Eds., volume 2034, Springer Berlin Heidelberg, 2001, pages 147–161, ISBN: 978-3-540-41866-5. DOI: 10.1007/3-540-45351-2_15 (cited on page 143).

[45] Y. Bertot and P. Castéran, *Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions*. New York, USA: Springer-Verlag, 2004, ISBN: 978-3-540-20854-9 (cited on page 82).

[46] A. Bianco and L. Alfaro, "Model checking of probabilistic and nondeterministic systems", in *Foundations of Software Technology and Theoretical Computer Science*, series Lecture Notes in Computer Science, P. Thiagarajan, Ed., volume 1026, Berlin, Heidelberg: Springer-Verlag, 1995, pages 499–513, ISBN: 978-3-540-60692-5. DOI: 10.1007/3-540-60692-0_70 (cited on pages 90, 91, 113, 253).

[47] A. Biere, E. Clarke, R. Raimi, and Y. Zhu, "Verifying safety properties of a PowerPC microprocessor using symbolic model checking without BDDs", in *Computer Aided Verification*, series Lecture Notes in Computer Science, N. Halbwachs and D. Peled, Eds., volume 1633, Springer Berlin Heidelberg, 1999, pages 60–71, ISBN: 978-3-540-66202-0. DOI: 10.1007/3-540-48683-6_8 (cited on pages 94, 98).

[48] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival, "The essence of computation", in, D. A. Mogensen Torben Æand Schmidt and I. H. Sudborough, Eds., New York, NY, USA: Springer-Verlag New York, Inc., 2002, ch. Design and Implementation of a Special-purpose Static Program Analyzer for Safety-critical Real-time Embedded Software, pages 85–108, ISBN: 3-540-00326-6. DOI: 10.1007/3-540-36377-7_5 (cited on page 88).

[49] C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. R. Nielson, "Static validation of security protocols", *Journal of Computer Security*, volume 13, number 3, pages 347–390, May 2005, ISSN: 0926-227X. DOI: 1145948.1145950 (cited on page 86).

[50] H. Bohnenkamp, H. Hermanns, J.-P. Katoen, and R. Klaren, "The modest modeling tool and its implementation", in *Computer Performance Evaluation. Modelling Techniques and Tools*, series Lecture Notes in Computer Science, P. Kemper and W. Sanders, Eds., volume 2794, Springer Berlin Heidelberg, 2003, pages 116–133, ISBN: 978-3-540-40814-7. DOI: 10.1007/978-3-540-45232-4_8 (cited on pages 100, 297, 299).

[51] M. P. Bonacina and J. Hsiang, "On the modelling of search in theorem proving - towards a theory of strategy analysis", *Information and Computation*, volume 147, number 2, pages 171–208, 1998. DOI: 10.1006/inco.1998.2739 (cited on page 82).

[52] C. A. Bond, C. L. Raehl, and T. Franke, "Medication errors in United States hospitals", *The Journal of Human Pharmacology and Drug Therapy*, volume 21, number 9, pages 1023–1036, Sep. 2001, ISSN: 0277-0008. DOI: 10.1592/phco.21.13.1023.34617 (cited on page 2).

[53] E. Börger, "Modeling workflow patterns from first principles", in *Conceptual Modeling - ER 2007*, series Lecture Notes in Computer Science, C. Parent, K.-D. Schewe, V. Storey, and B. Thalheim, Eds., volume 4801, Berlin, Heidelberg: Springer-Verlag, 2007, pages 1–20, ISBN: 978-3-540-75562-3. DOI: 10.1007/978-3-540-75563-0_1 (cited on page 30).

[54] E. Börger, "Approaches to modeling business processes: a critical analysis of BPMN, workflow patterns and YAWL", *Software and Systems Modeling (SoSyM)*, volume 11, number 3, pages 305–318, Jul. 2012, ISSN: 1619-1366. DOI: 10.1007/s10270-011-0214-z (cited on pages 27, 33, 155).

[55] E. Börger and O. Sörensen, "BPMN core modeling concepts: inheritance-based execution semantics", English, in *Handbook of Conceptual Modeling*, D. W. Embley and B. Thalheim, Eds., Berlin, Heidelberg: Springer-Verlag, 2011, pages 287–332, ISBN: 978-3-642-15864-3. DOI: 10.1007/978-3-642-15865-0_9 (cited on page 27).

[56] E. Börger and B. Thalheim, "A method for verifiable and validatable business process modeling", in *Advances in Software Engineering*, series Lecture Notes in Computer Science, E. Börger and A. Cisternino, Eds., volume 5316, Springer Berlin Heidelberg, 2008, pages 59–115, ISBN: 978-3-540-89761-3. DOI: 10.1007/978-3-540-89762-0_3 (cited on pages 27, 62).

[57] K. R. Braghetto, J. E. Ferreira, and J.-M. Vincent, "From business process model and notation to stochastic automata network", University of São Paulo, Institute of Mathematics and statistics, department of computer science, Sao Paulo, Brazil, Specification RT-MAC-2011-03, Mar. 2011. [Online]. Available: http://www.ime.usp.br/~kellyrb/files/fromBPMNtoSAN(technical_report).pdf (cited on page 123).

[58] K. R. Braghetto, J. E. Ferreira, and J.-M. Vincent, "Performance evaluation of business processes through a formal transformation to san", in *Computer Performance Engineering*, series Lecture Notes in Computer Science, N. Thomas, Ed., volume 6977, Springer Berlin Heidelberg, 2011, pages 42–56, ISBN: 978-3-642-24748-4. DOI: 10.1007/978-3-642-24749-1_5 (cited on page 123).

[59] L. Brenner, P. Fernandes, B. Plateau, and I. Sbeity, "PEPS 2007 - stochastic automata networks software tool", in *Fourth International Conference on the Quantitative Evaluation of Systems, 2007 (QEST 2007)*, Sep. 2007, pages 163–164. DOI: 10.1109/QEST.2007.33 (cited on pages 123, 297).

[60] P. J. Broadfoot and A. W. Roscoe, "Tutorial on FDR and its applications", in *Proceedings of the 7th International SPIN Workshop on SPIN Model Checking and Software Verification*, London, UK: Springer-Verlag, 2000, pages 322–, ISBN: 3-540-41030-9. [Online]. Available: http://dl.acm.org/citation.cfm?id=645880.672219 (cited on pages 92, 119, 295).

[61] M. Bromwich and C. Hong, "Activity-based costing systems and incremental costs", *Management Accounting Research*, volume 10, number 1, pages 39–60, 1999, ISSN: 1044-5005. DOI: 10.1006/mare.1998.0102 (cited on page 18).

[62] R. Bryant, "Graph-based algorithms for boolean function manipulation", *IEEE Transactions on Computers*, volume C-35, number 8, pages 677–691, Aug. 1986, ISSN: 0018-9340. DOI: 10.1109/TC.1986.1676819 (cited on pages 97, 297).

[63] P. Bulychev, A. David, K. Guldstrand Larsen, A. Legay, M. Mikučionis, and D. Bøgsted Poulsen, "Checking and distributing statistical model checking", in *NASA Formal Methods*, series Lecture Notes in Computer

Science, A. E. Goodloe and S. Person, Eds., volume 7226, Springer Berlin Heidelberg, 2012, pages 449–463, ISBN: 978-3-642-28890-6. DOI: 10.1007/978-3-642-28891-3_39 (cited on pages 100, 297).

[64] J. R. Burch, E. Clarke, K. L. McMillan, D. Dill, and L. J. Hwang, "Symbolic model checking: 10^{20} states and beyond", in *Logic in Computer Science, 1990. LICS '90, Proceedings., Fifth Annual IEEE Symposium on e*, Jun. 1990, pages 428–439. DOI: 10.1109/LICS.1990.113767 (cited on page 97).

[65] T. Burkhart, J. Krumeich, D. Werth, and P. Loos, "Analyzing the business model concept - a comprehensive classification of literature", in *Proceedings of the International Conference on Information Systems, ICIS 2011, Shanghai, China, December 4-7, 2011*, D. F. Galletta and T.-P. Liang, Eds., Association for Information Systems, 2011, ISBN: 978-0-615-55907-0 (cited on pages 14, 15, 40).

[66] C. Canevet, S. Gilmore, J. Hillston, M. Prowse, and P. Stevens, "Performance modelling with the unified modelling language and stochastic process algebras", *IEE Proceedings - Computers and Digital Techniques*, volume 150, number 2, pages 107–120, Mar. 2003, ISSN: 1350-2387. DOI: 10.1049/ip-cdt:20030084 (cited on pages 26, 33).

[67] J. Carmenates and M. R. Keith, "Impact of automation on pharmacist interventions and medication errors in a correctional health care system", *American Journal Of Health-system Pharmacy*, volume 59, number 9, pages 779–783, May 2001, ISSN: 1079-2082. DOI: 10.1136/archdischild-2011-301239 (cited on pages 2, 3).

[68] P. Černý, K. Chatterjee, T. A. Henzinger, A. Radhakrishna, and R. Singh, "Quantitative synthesis for concurrent programs", in *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV'11)*, G. Gopalakrishnan and S. Qadeer, Eds., series Lecture Notes in Computer Science, volume 6806, London, UK: Springer-Verlag, 2011, pages 243–259, ISBN: 978-3-642-22109-5. DOI: 10.1007/978-3-642-22110-1_20 (cited on page 196).

[69] K. Chatterjee, T. A. Henzinger, and B. Jobstmann, "Environment assumptions for synthesis", in *CONCUR 2008 - Concurrency Theory*, series Lecture Notes in Computer Science, F. Breugel and M. Chechik, Eds., volume 5201, Berlin, Heidelberg: Springer-Verlag, 2008, pages 147–161, ISBN: 978-3-540-85360-2. DOI: 10.1007/978-3-540-85361-9_14 (cited on page 241).

[70] T. Chen, V. Forejt, M. Kwiatkowska, D. Parker, and A. Simaitis, "Automatic verification of competitive stochastic systems", in *Proceedings of the 18th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'12)*, C. Flanagan and B. König,

Eds., series Lecture Notes in Computer Science, volume 7214, Berlin, Heidelberg: Springer-Verlag, 2012, pages 315–330. DOI: 10.1007/978-3-642-28756-5_22 (cited on pages 239, 240).

[71] M. Chinosi and A. Trombetta, "BPMN: an introduction to the standard", *Computer Standards & Interfaces*, volume 34, number 1, pages 124–134, Jan. 2012, ISSN: 0920-5489. DOI: 10.1016/j.csi.2011.06.002 (cited on page 48).

[72] D. R. Christiansen, M. Carbone, and T. Hildebrandt, "Formal semantics and implementation of BPMN 2.0 inclusive gateways", in *Proceedings of the 7th international conference on Web services and formal methods*, M. Bravetti and T. Bultan, Eds., series Lecture Notes in Computer Science, volume 6551, Berlin, Heidelberg: Springer-Verlag, 2011, pages 146–160, ISBN: 978-3-642-19588-4. DOI: 10.1007/978-3-642-19589-1_10 (cited on pages 27, 64).

[73] A. Church, "A set of postulates for the foundation of logic", *Annals of mathematics*, volume 33, number 2, pages 346–366, 1932 (cited on page 68).

[74] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "NuSMV 2: an opensource tool for symbolic model checking", in *Computer Aided Verification*, E. Brinksma and K. Larsen, Eds., series Lecture Notes in Computer Science, volume 2404, London, UK, UK: Springer-Verlag, 2002, pages 359–364, ISBN: 978-3-540-43997-4. DOI: 10.1007/3-540-45657-0_29 (cited on pages 26, 297).

[75] E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic verification of finite-state concurrent systems using temporal logic specifications", *ACM Transactions on Programming Languages and Systems*, volume 8, number 2, pages 244–263, 1986, ISSN: 0164-0925. DOI: 10.1145/5397.5399 (cited on pages 89, 90, 247, 248).

[76] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Counterexample-guided abstraction refinement for symbolic model checking", *Journal of the ACM*, volume 50, number 5, pages 752–794, Sep. 2003, ISSN: 0004-5411. DOI: 10.1145/876638.876643 (cited on pages 98, 295).

[77] E. Clarke, R. Enders, T. Filkorn, and S. Jha, "Exploiting symmetry in temporal logic model checking", *Formal Methods in System Design*, volume 9, number 1-2, pages 77–104, 1996, ISSN: 0925-9856. DOI: 10.1007/BF00625969 (cited on page 98).

[78] E. Clarke, M. Khaira, and X. Zhao, "Word level model checking-avoiding the pentium FDIV error", in *Design Automation Conference Proceedings 1996, 33rd*, Jun. 1996, pages 645–648. DOI: 10.1109/DAC.1996.545654 (cited on page 94).

[79] A. Colquhoun, "Could automation improve efficieny and help pharmacies with cost saving?", *The Pharmaceutical Journal*, volume 285, pages 587–591, Nov. 2010, ISSN: 0031-687 (cited on page 2).

[80] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*, 3rd. The MIT Press, 2009, ISBN: 0262033844 (cited on pages 168, 187, 208).

[81] P. Cousot and R. Cousot, "Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints", in *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, series POPL '77, New York, NY, USA: ACM, 1977, pages 238–252. DOI: 10.1145/512950.512973 (cited on page 86).

[82] P. Cousot and R. Cousot, "Comparing the Galois connection and widening/narrowing approaches to abstract interpretation", in *Programming Language Implementation and Logic Programming*, series Lecture Notes in Computer Science, M. Bruynooghe and M. Wirsing, Eds., volume 631, Springer Berlin Heidelberg, 1992, pages 269–295, ISBN: 978-3-540-55844-6. DOI: 10.1007/3-540-55844-6_142 (cited on page 87).

[83] N. Crockford, *An Introduction to Risk Management*, 2nd Edition. Cambridge England: Woodhead-Faulkner, 1986, ISBN: 9780859413329 (cited on page 155).

[84] B. Curtis, M. I. Kellner, and J. Over, "Process modeling", *Communications of the ACM*, volume 35, number 9, pages 75–90, Sep. 1992, ISSN: 0001-0782. DOI: 10.1145/130994.130998 (cited on page 22).

[85] V. Danos, J. Feret, W. Fontana, R. Harmer, J. Krivine, P. Biosystems, É. N. Supérieure, and É. Polytechnique, "Rule-based modelling of cellular signalling", in *Proceedings of the 18th International Conference on Concurrency Theory (CONCUR'07)*, series Lecture Notes in Computer Science, 2007, pages 17–41. DOI: 10.1.1.107.228 (cited on page 237).

[86] M. Daumas, L. Rideau, and L. Théry, "A generic library for floating-point numbers and its application to exact computing", English, in *Theorem Proving in Higher Order Logics*, series Lecture Notes in Computer Science, R. J. Boulton and P. B. Jackson, Eds., volume 2152, Springer Berlin Heidelberg, 2001, pages 169–184, ISBN: 978-3-540-42525-0. DOI: 10.1007/3-540-44755-5_13 (cited on page 95).

[87] T. H. Davenport, *Process innovation: reengineering work through information technology*. Boston, MA, USA: Harvard Business School Press, 1993, ISBN: 0-87584-366-2 (cited on pages 18, 19).

[88] R. M. Dijkman, M. Dumas, and C. Ouyang, "Semantics and analysis of business process models in BPMN", *Information and Software Technology*, volume 50, number 12, pages 1281–1294, 12 Nov. 2008, ISSN: 0950-5849. DOI: 10.1016/j.infsof.2008.02.006 (cited on pages 27, 28, 33, 49, 119, 120).

[89] G. L. Dodaro and B. P. Crowley, "Business process reengineering assessment guide", U.S. Government Accountability Office, GAO AIMD-10.1.15, 1997. [Online]. Available: http://www.gao.gov/assets/80/76302.pdf (cited on pages 177, 199).

[90] A. Donaldson, A. Miller, and D. Parker, "Language-level symmetry reduction for probabilistic model checking", in *Quantitative Evaluation of Systems, 2009. QEST '09. Sixth International Conference on the*, Sep. 2009, pages 289–298. DOI: 10.1109/QEST.2009.21 (cited on page 104).

[91] B. F. Dongen, A. K. A. Medeiros, H. M. W. Verbeek, A. J. M. M. Weijters, and W. M. P. Aalst, "The prom framework: a new era in process mining tool support", in *Applications and Theory of Petri Nets 2005*, series Lecture Notes in Computer Science, G. Ciardo and P. Darondeau, Eds., volume 3536, Berlin, Heidelberg: Springer, 2005, pages 444–454, ISBN: 978-3-540-26301-2. DOI: 10.1007/11494744_25 (cited on page 119).

[92] M. Dumas, A. Grosskopf, T. Hettel, and M. Wynn, "Semantics of standard process models with OR-joins", in *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS*, series Lecture Notes in Computer Science, R. Meersman and Z. Tari, Eds., volume 4803, Springer Berlin Heidelberg, 2007, pages 41–58, ISBN: 978-3-540-76846-3. DOI: 10.1007/978-3-540-76848-7_5 (cited on pages 27, 64).

[93] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett, "Patterns in property specifications for finite-state verification", in *Proceedings of the 21st International Conference on Software Engineering*, series ICSE '99, New York, NY, USA: ACM, 1999, pages 411–420, ISBN: 1-58113-074-0. DOI: 10.1145/302405.302672 (cited on page 120).

[94] E. Emerson and E. Clarke, "Characterizing correctness properties of parallel programs using fixpoints", in *Automata, Languages and Programming*, series Lecture Notes in Computer Science, J. de Bakker and J. van Leeuwen, Eds., volume 85, Berlin, Heidelberg: Springer-Verlag, 1980, pages 169–181. DOI: 10.1007/3-540-10003-2_69 (cited on page 89).

[95] E. Emerson and E. M. Clarke, "Using branching time temporal logic to synthesize synchronization skeletons", *Science of Computer Programming*, volume 2, number 3, pages 241–266, 1982, ISSN: 0167-6423. DOI: 10.1016/0167-6423(83)90017-5 (cited on page 90).

[96] C. A. Ericson, "Fault tree analysis - a history", in *Proceedings of the 17th International System Safety Conference*, series ISSC'99, Unionville, Virginia, USA: System Safety Society, 1999, pages 1–9 (cited on page 156).

[97] C. A. Ericson, "Fault tree analysis", in *Hazard Analysis Techniques for System Safety*, C. A. Ericson, Ed., New Jersey, USA: John Wiley & Sons, Inc., 2005, pages 183–221, ISBN: 9780471739425. DOI: 10.1002/0471739421.ch11 (cited on pages 5, 155, 156, 158, 159, 295).

[98] R. Eshuis, "Symbolic model checking of UML activity diagrams", *ACM Transactions on Software Engineering and Methodology*, volume 15, number 1, pages 1–38, Jan. 2006, ISSN: 1049-331X. DOI: 10.1145/1125808.1125809 (cited on pages 26, 33).

[99] M. E. Fagan, "Design and code inspections to reduce errors in program development", *IBM Systems Journal*, volume 15, number 3, pages 182–211, 1976, ISSN: 0018-8670. DOI: 10.1147/sj.153.0182 (cited on page 17).

[100] C. Favre and H. Völzer, "The difficulty of replacing an inclusive OR-join", in *Business Process Management*, series Lecture Notes in Computer Science, A. Barros, A. Gal, and E. Kindler, Eds., volume 7481, Berlin, Heidelberg: Springer-Verlag, 2012, pages 156–171, ISBN: 978-3-642-32884-8. DOI: 10.1007/978-3-642-32885-5_12 (cited on page 64).

[101] H. Fecher, J. Schönborn, M. Kyas, and W.-P. Roever, "29 New unclarities in the semantics of UML 2.0 State machines", in *Formal Methods and Software Engineering*, series Lecture Notes in Computer Science, K.-K. Lau and R. Banach, Eds., volume 3785, Berlin, Heidelberg: Springer-Verlag, 2005, pages 52–65, ISBN: 978-3-540-29797-0. DOI: 10.1007/11576280_5 (cited on page 25).

[102] J. Ferreiros, "The road to modern logic - an interpretation", *The Bulletin of Symbolic Logic*, volume 7, number 4, pages 441–484, 2001. [Online]. Available: http://www.jstor.org/stable/2687794 (cited on pages 79, 80).

[103] Food and Drug Administration, "Quality system regulation", FDA 21 CFR 820, 2011 (cited on page 157).

[104] V. Forejt, M. Kwiatkowska, G. Norman, D. Parker, and H. Qu, "Quantitative multi-objective verification for probabilistic systems", in *Tools and Algorithms for the Construction and Analysis of Systems*, series Lecture Notes in Computer Science, P. A. Abdulla and K. M. Leino, Eds., volume 6605, Springer Berlin Heidelberg, 2011, pages 112–127, ISBN: 978-3-642-19834-2. DOI: 10.1007/978-3-642-19835-9_11 (cited on page 144).

[105] V. Forejt, M. Kwiatkowska, and D. Parker, "Pareto curves for probabilistic model checking", in *Automated Technology for Verification and Analysis*, series Lecture Notes in Computer Science, S. Chakraborty and M. Mukund, Eds., Springer Berlin Heidelberg, 2012, pages 317–332, ISBN: 978-3-642-33385-9. DOI: 10.1007/978-3-642-33386-6_25 (cited on pages 144, 146).

[106] R. France, S. Ghosh, T. Dinh-Trong, and A. Solberg, "Model-driven development using UML 2.0: promises and pitfalls", *Computer*, volume 39, number 2, pages 59–66, 2006, ISSN: 0018-9162. DOI: 10.1109/MC.2006.65 (cited on page 25).

[107] J. Frijters. (Dec. 2013). Ikvm.net website, [Online]. Available: http://www.ikvm.net/ (cited on pages 203, 205).

[108] M. P. Gallaher and B. M. Kropp, "The economic impacts of inadequate infrastructure for software testing", NIST, Gaithersburg MD, USA, Planning Report 7007.011, May 2002. [Online]. Available: http://www.nist.gov/director/prog-ofc/report02-3.pdf (cited on pages 17, 45).

[109] X. Ge, R. F. Paige, and J. A. McDermid, "Analysing system failure behaviours with PRISM", in *Fourth International Conference on Secure Software Integration and Reliability Improvement Companion (SSIRI-C)*, Jun. 2010, pages 130–136. DOI: 10.1109/SSIRI-C.2010.32 (cited on page 161).

[110] A. Gelman, D. K. Park, S. Ansolabehere, P. N. Price, and L. C. Minnite, "Models, assumptions and model checking in ecological regressions", *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, volume 164, number 1, pages 101–118, 2001, ISSN: 1467-985X. DOI: 10.1111/1467-985X.00190 (cited on page 91).

[111] H. Gibson, J. Faith, and P. Vickers, "A survey of two-dimensional graph layout techniques for information visualisation", *Information Visualization*, 2012. DOI: 10.1177/1473871612455749 (cited on page 149).

[112] C. Gilbert and J. L. Bower, "Disruptive change: when trying harder is part of the problem", *Harvard Business Review*, volume 80, number 5, pages 94–101, May 2002, ISSN: 0017-8012. DOI: 10.1225/9993 (cited on pages 17, 176).

[113] F. B. Gilbreth and L. M. Gilbreth, *Process Charts - First Steps in Finding the One Best Way*. New York, NY: American Society of Mechanical Engineers (ASME), 1921, ISBN: B00088R3U4 (cited on pages 15, 21, 22, 176, 238).

[114] S. Giro and M. N. Rabe, "Verification of partial-information probabilistic systems using counterexample-guided refinements", in *Automated Technology for Verification and Analysis*, series Lecture Notes in Computer Science, S. Chakraborty and M. Mukund, Eds., Springer Berlin Heidelberg, 2012, pages 333–348, ISBN: 978-3-642-33385-9. DOI: 10.1007/978-3-642-33386-6_26 (cited on page 229).

[115] F. Giunchiglia and P. Traverso, "Planning as model checking", in *Recent Advances in AI Planning*, series Lecture Notes in Computer Science, S. Biundo and M. Fox, Eds., volume 1809, Springer Berlin Heidelberg, 2000, pages 1–20, ISBN: 978-3-540-67866-3. DOI: 10.1007/10720246_1 (cited on page 143).

[116] P. Godefroid, "Using partial orders to improve automatic verification methods", in *Computer-Aided Verification*, series Lecture Notes in Computer Science, E. M. Clarke and R. P. Kurshan, Eds., volume 531, Springer Berlin Heidelberg, 1991, pages 176–185, ISBN: 978-3-540-54477-7. DOI: 10.1007/BFb0023731 (cited on page 98).

[117] S. Goedertier and J. Vanthienen, "Designing compliant business processes with obligations and permissions", in *BPM 2006 Workshops*, series LNCS, J. Eder and S. Dustdar, Eds., volume 4103, Berlin, Heidelberg: Springer-Verlag, 2006, pages 5–14, ISBN: 978-3-540-38444-1. DOI: 10.1007/11837862_2 (cited on page 181).

[118] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, 1st. Boston, MA, USA: Addison-Wesley, 1989, ISBN: 0201157675 (cited on pages 183, 186, 187, 198).

[119] M. Größer, G. Norman, C. Baier, F. Ciesinski, M. Kwiatkowska, and D. Parker, "On reduction criteria for probabilistic reward models", in *Proceedings of the 26th international conference on Foundations of Software Technology and Theoretical Computer Science*, series FSTTCS'06, Berlin, Heidelberg: Springer-Verlag, 2006, pages 309–320. DOI: 10.1007/11944836_29 (cited on page 209).

[120] A. Grosskopf, "xBPMN - Formal Control Flow Specification of a BPMN based Process Execution Language", Master's thesis, Hasso Plattner Institute for IT Systems Engineering, Potsdam, Germany, Jul. 2007. [Online]. Available: http://bpt.hpi.uni-potsdam.de/pub/Public/AlexanderGrosskopf/xBPMN_thesis.pdf (cited on page 27).

[121] V. Gruhn and R. Laue, "Complexity metrics for business process models", in *9th International Conference on Business Information Systems (BIS 2006)*, H. C. M. Witold Abramowicz, Ed., series Lecture Notes in Informatics, Springer-Verlag, 2006, pages 1–12, ISBN: 978-3-540-72034-8 (cited on pages 59, 60).

[122] V. Gruhn and R. Laue, "What business process modelers can learn from programmers", *Science of Computer Programming*, volume 65, number 1, pages 4–13, Mar. 2007, ISSN: 0167-6423. DOI: 10.1016/j.scico.2006.08.003 (cited on pages 60, 64, 225).

[123] S. Guha, W. J. Kettinger, and J. T. Teng, "Business process reengineering: building a comprehensive methodology", *Information Systems Management*, volume 10, number 3, pages 13–22, 1993. DOI: 10.1080/10580539308906939 (cited on pages 178, 294).

[124] M. Hague and C.-H. L. Ong, "Analysing mu-calculus properties of pushdown systems", in *Proceedings of the 17th International SPIN Conference on Model Checking Software*, series SPIN'10, Berlin, Heidelberg: Springer-Verlag, 2010, pages 187–192, ISBN: 3-642-16163-4, 978-3-642-16163-6. [Online]. Available: http://dl.acm.org/citation.cfm?id=1928137.1928156 (cited on page 86).

[125] M. Hamilton, *Software Development: Building Reliable Systems*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1999, ISBN: 0-13-081246-3 (cited on page 25).

[126] M. Hammer, "Reengineering work: don't automate, obliterate", *Harvard Business Review*, volume 68, number 4, pages 104–112, 1990, ISSN: 00178012. DOI: 10.1225/90406 (cited on pages 18, 19, 176, 177, 294).

[127] W. Hamscher, "AI in business-process reengineering", *AI Magazine*, volume 15, number 4, pages 71–72, 1994. DOI: 10.1609/aimag.v15i4.1113 (cited on page 177).

[128] P. Y. Han, I. D. Coombes, and B. Green, "Factors predictive of intravenous fluid administration errors in Australian surgical care wards", *Quality and safety in health care*, volume 14, pages 179–184, 2004, ISSN: 1475-3898. DOI: 10.1136/qshc.2004.010728 (cited on pages 2, 3).

[129] H. Hansen, M. Kwiatkowska, and H. Qu, "Partial order reduction for model checking Markov decision processes under unconditional fairness", in *Quantitative Evaluation of Systems (QEST), 2011 Eighth International Conference on*, Sep. 2011, pages 203–212. DOI: 10.1109/QEST.2011.35 (cited on page 104).

[130] S. M. Hansen, J. Skriver, and H. R. Nielson, "Using static analysis to validate the saml single sign-on protocol", in *Proceedings of the 2005 Workshop on Issues in the Theory of Security*, series WITS '05, New York, NY, USA: ACM, 2005, pages 27–40, ISBN: 1-58113-980-2. DOI: 10.1145/1045405.1045409 (cited on page 86).

[131] H. Hansson and B. Jonsson, "A logic for reasoning about time and reliability", *Formal Aspects of Computing*, volume 6, number 5, pages 512–535, 1994, ISSN: 0934-5043. DOI: 10.1007/BF01211866 (cited on pages 90, 113, 247–249, 297).

[132] J. Harrison, *Handbook of Practical Logic and Automated Reasoning*, 1st. New York, NY, USA: Cambridge University Press, 2009, ISBN: 0521899575 (cited on pages 80, 82).

[133] J. Heath, M. Kwiatkowska, G. Norman, D. Parker, and O. Tymchyshyn, "Probabilistic model checking of complex biological pathways", *Theoretical Computer Science*, volume 319, number 3, pages 239–257, 2008, ISSN: 0304-3975. DOI: 10.1016/j.tcs.2007.11.013 (cited on pages 91, 94).

[134] J. B. Hill, M. Pezzini, and Y. V. Natis, "Findings: confusion remains regarding BPM terminologies", *Gartner Reports*, number G00155817, 2008 (cited on page 18).

[135] J. Hillston, *A compositional approach to performance modelling*. New York, NY, USA: Cambridge University Press, 1996, ISBN: 0-521-57189-8 (cited on pages 26, 297).

[136] B. L. Hintzen, S. J. Knoer, C. J. V. Dyke, and B. S. Milavitz, "Effect of LEAN process improvement techniques on a university hospital inpatient pharmacy", *American Journal Of Health-system Pharmacy*, volume 66, number 22, pages 2042–2047, Nov. 2009, ISSN: 1079-2082. DOI: 10.2146/ajhp080540 (cited on page 2).

[137] C. A. R. Hoare, "Communicating sequential processes", *Communications of the ACM*, volume 21, number 8, pages 666–677, Aug. 1983, ISSN: 0001-0782. DOI: 10.1145/359576.359585 (cited on pages 23, 67, 294).

[138] C. A. R. Hoare, *Communicating Sequential Processes*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1985, ISBN: 0-13-153271-5 (cited on pages 23, 67, 105, 111, 119).

[139] A. Hofstede, W. Aalst, M. Adams, and N. Russell, *Modern Business Process Automation: YAWL and its Support Environment*. Berlin, Heidelberg: Springer-Verlag, 2010, ISBN: 978-3-642-03120-5 (cited on pages 29, 30).

[140] G. Holzmann, *Spin model checker, the: primer and reference manual*, 1st. Addison-Wesley Professional, 2003, ISBN: 0-321-22862-6 (cited on pages 91, 143, 298).

[141] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2006, ISBN: 0321455363 (cited on page 84).

[142] M. Huth and M. Kwiatkowska, "Quantitative analysis and model checking", in *Logic in Computer Science, 1997. LICS '97. Proceedings., 12th Annual IEEE Symposium on*, Jun. 1997, pages 111–122. DOI: 10.1109/LICS.1997.614940 (cited on pages 44, 61).

[143] International Organization for Standardization, "Medical devices: quality management systems - requirements for regulatory purposes", ISO 13485, 2003 (cited on page 157).

[144] International Organization for Standardization, "Medical devices - application of risk management to medical devices", ISO 14971, 2007 (cited on page 157).

[145] International Telecommunication Union, "Specification and description language (SDL)", ITU, Geneva, Switzerland, Specification Z.100-Z.1091, Nov. 1999. [Online]. Available: http://www.itu.int/ITU-T/studygroups/com10/languages/Z.100_1199.pdf (cited on page 237).

[146] K. Ishikawa, *What is total quality control? The Japanese way*, series Prentice Hall business classics. Prentice-Hall, 1985 (cited on page 18).

[147] T. Jaeger, A. Prakash, and M. Ishikawa, "A framework for automatic improvement of workflows to meet performance goals", in *Proceedings of the Sixth International Conference on Tools with Artificial Intelligence*, Nov. 1994, pages 640–646. DOI: 10.1109/TAI.1994.346434 (cited on page 182).

[148] D. N. Jansen, H. Hermanns, and J.-P. Katoen, "A probabilistic extension of UML statecharts", in *Proceedings of the 7th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems: Co-sponsored by IFIP WG 2.2*, W. Damm and E.-R. Olderog, Eds., series FTRTFT '02, volume 2469, London, UK, UK: Springer, 2002, pages 355–374, ISBN: 3-540-44165-4. DOI: 646847.707117 (cited on pages 26, 33, 226).

[149] K. Jensen, *Coloured Petri Nets (2Nd Ed.): Basic Concepts, Analysis Methods and Practical Use: Volume 1*. London, UK, UK: Springer-Verlag, 1996, ISBN: 3-540-60943-1 (cited on page 30).

[150] F. Kamrani, R. Ayani, and A. Karimson, "Optimizing a business process model by using simulation", in *Principles of Advanced and Distributed Simulation (PADS), 2010 IEEE Workshop on*, IEEE, May 2010, pages 1–8. DOI: 10.1109/PADS.2010.5471671 (cited on page 182).

[151] J.-P. Katoen, I. S. Zapreev, E. M. Hahn, H. Hermanns, and D. N. Jansen, "The ins and outs of the probabilistic model checker MRMC", in *Proceedings of the 2009 Sixth International Conference on the Quantitative Evaluation of Systems*, series QEST '09, Washington, DC, USA: IEEE Computer Society, 2009, pages 167–176, ISBN: 978-0-7695-3808-2. DOI: 10.1109/QEST.2009.11 (cited on pages 100, 297).

[152] M. Kattenbelt, M. Kwiatkowska, G. Norman, and D. Parker, "Abstraction refinement for probabilistic software", in *Verification, Model Checking, and Abstract Interpretation*, N. D. Jones and M. Müller-Olm, Eds., series Lecture Notes in Computer Science, volume 5403, Berlin, Heidelberg: Springer-Verlag, 2009, pages 182–197, ISBN: 978-3-540-93899-6. DOI: 10.1007/978-3-540-93900-9_17 (cited on page 104).

[153] J. G. Kemeny, J. L. Snell, and A. W. Knapp, *Denumerable Markov chains (second edition)*. Springer, 1976 (cited on pages 245, 247).

[154] B. Kiepuszewski, "Expressiveness and suitability of languages for control flow modelling in workflows", PhD thesis, Queensland University of Technology, Brisbane, Australia, 2003. [Online]. Available: http://eprints.qut.edu.au/36882/ (cited on pages 24, 29).

[155] B. Kiepuszewski, A. H. M. t. Hofstede, and C. Bussler, "On structured workflow modelling", in *Proceedings of the 12th International Conference on Advanced Information Systems Engineering*, B. Wangler and L. Bergman, Eds., series Lecture Notes in Computer Science, volume 1789, Berlin, Heidelberg: Springer-Verlag, 2000, pages 431–445, ISBN: 978-3-540-67630-0. DOI: 10.1007/3-540-45140-4_29 (cited on page 57).

[156] G. A. Kildall, "A unified approach to global program optimization", in *Proceedings of the 1st Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, series POPL '73, New York, NY, USA: ACM, 1973, pages 194–206. DOI: 10.1145/512927.512945 (cited on page 86).

[157] J. C. King, "Symbolic execution and program testing", *Communications of the ACM*, volume 19, number 7, pages 385–394, Jul. 1976, ISSN: 0001-0782. DOI: 10.1145/360248.360252 (cited on page 86).

[158] M. Klein and C. Dellarocas, "Designing robust business processes", in *In Organizing Business Knowledge: The MIT Process*, MIT Press, 2003, pages 434–438 (cited on page 16).

[159] R. K. L. Ko, "A computer scientist's introductory guide to business process management (BPM)", *ACM Crossroads*, volume 15, number 4, 4:11–4:18, Jun. 2009, ISSN: 1528-4972. DOI: 10.1145/1558897.1558901 (cited on pages 20, 294).

[160] S. Konur, C. Dixon, and M. Fisher, "Analysing robot swarm behaviour via probabilistic model checking", *Robotics and Autonomous Systems*, volume 60, number 2, pages 199–213, 2012, ISSN: 0921-8890. DOI: 10.1016/j.robot.2011.10.005 (cited on page 91).

[161] J. Kotter, *Leading Change*. Harvard Business School Press, 1996, ISBN: 9780875847474 (cited on pages 142, 176).

[162] Y. Kouskoulas, D. Renshaw, A. Platzer, and P. Kazanzides, "Certifying the safe design of a virtual fixture control algorithm for a surgical robot", in *Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control*, series HSCC '13, Philadelphia, Pennsylvania, USA: ACM, 2013, pages 263–272, ISBN: 978-1-4503-1567-8. DOI: 10.1145/2461328.2461369 (cited on page 95).

[163] L. Kovács and A. Voronkov, "First-order theorem proving and vampire", in *Computer Aided Verification*, series Lecture Notes in Computer Science, N. Sharygina and H. Veith, Eds., volume 8044, Springer Berlin Heidelberg, 2013, pages 1–35, ISBN: 978-3-642-39798-1. DOI: 10.1007/978-3-642-39799-8_1 (cited on page 82).

[164] D. Kozen, "Results on the propositional μ-calculus", in *Automata, Languages and Programming*, series Lecture Notes in Computer Science, M. Nielsen and E. Schmidt, Eds., volume 140, Springer Berlin Heidelberg, 1982, pages 348–359, ISBN: 978-3-540-11576-2. DOI: 10.1007/BFb0012782 (cited on page 239).

[165] M. Kunze, M. Weidlich, and M. Weske, "Behavioral similarity - a proper metric", in *Business Process Management*, series Lecture Notes in Computer Science, S. Rinderle-Ma, F. Toumani, and K. Wolf, Eds., volume 6896, Berlin, Heidelberg: Springer-Verlag, 2011, pages 166–181, ISBN: 978-3-642-23058-5. DOI: 10.1007/978-3-642-23059-2_15 (cited on pages 235, 240).

[166] M. Kwiatkowska, G. Norman, and D. Parker, "Probabilistic symbolic model checking with PRISM: a hybrid approach", *International Journal on Software Tools for Technology Transfer*, volume 6, number 2, pages 128–142, Aug. 2004, ISSN: 1433-2779. DOI: 10.1007/s10009-004-0140-2 (cited on page 102).

[167] M. Kwiatkowska, G. Norman, and D. Parker, "Symmetry reduction for probabilistic model checking", in *Proceedings of 18th International Conference on Computer Aided Verification (CAV'06)*, T. Ball and R. Jones, Eds., series Lecture Notes in Computer Science, volume 4114, London, UK: Springer-Verlag, 2006, pages 234–248. DOI: 10.1109/QEST.2009.21 (cited on pages 127, 209).

[168] M. Kwiatkowska, G. Norman, and D. Parker, "Stochastic model checking", in *Formal Methods for Performance Evaluation*, series Lecture Notes in Computer Science, M. Bernardo and J. Hillston, Eds., volume 4486, Berlin, Heidelberg: Springer-Verlag, 2007, pages 220–270, ISBN: 978-3-540-72482-7. DOI: 10.1007/978-3-540-72522-0_6 (cited on page 249).

[169] M. Kwiatkowska, G. Norman, and D. Parker, "A framework for verification of software with time and probabilities", in *Formal Modeling and Analysis of Timed Systems*, series Lecture Notes in Computer Science, K. Chatterjee and T. A. Henzinger, Eds., volume 6246, Springer Berlin Heidelberg, 2010, pages 25–45, ISBN: 978-3-642-15296-2. DOI: 10.1007/978-3-642-15297-9_4 (cited on page 102).

[170] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: verification of probabilistic real-time systems", in *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV'11)*, G. Gopalakrishnan and S. Qadeer, Eds., series Lecture Notes in Computer Science,

volume 6806, London, UK: Springer-Verlag, 2011, pages 585–591, ISBN: 978-3-642-22109-5. DOI: 10.1007/978-3-642-22110-1_47 (cited on pages 91, 100, 122, 298).

[171] M. Kwiatkowska and D. Parker, "Advances in probabilistic model checking", in *Software Safety and Security - Tools for Analysis and Verification*, T. Nipkow, O. Grumberg, and B. Hauptmann, Eds., series NATO Science for Peace and Security Series - D: Information and Communication Security, volume 33, IOS Press, 2012, pages 126–151. DOI: 10.3233/978-1-61499-028-4-126 (cited on pages 104, 244).

[172] A. H. Land and A. G. Doig, "An automatic method for solving discrete programming problems", English, in *50 Years of Integer Programming 1958-2008*, M. Jünger, T. M. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, and L. A. Wolsey, Eds., Berlin, Heidelberg: Springer-Verlag, 2010, pages 105–132, ISBN: 978-3-540-68274-5. DOI: 10.1007/978-3-540-68279-0_5 (cited on page 143).

[173] A. Lapadula, R. Pugliese, and F. Tiezzi, "A calculus for orchestration of web services", in *Programming Languages and Systems*, series Lecture Notes in Computer Science, R. Nicola, Ed., volume 4421, Berlin, Heidelberg: Springer-Verlag, 2007, pages 33–47, ISBN: 978-3-540-71314-2. DOI: 10.1007/978-3-540-71316-6_4 (cited on pages 29, 49).

[174] P. Liggesmeyer and M. Rothfelder, "Improving system reliability with automatic fault tree generation", in *Fault-Tolerant Computing, 1998. Digest of Papers. Twenty-Eighth Annual International Symposium on*, Jun. 1998, pages 90–99. DOI: 10.1109/FTCS.1998.689458 (cited on pages 159, 161, 162).

[175] J. Lilius and I. P. Paltor, "Formalising UML state machines for model checking", in *≪UML≫'99 - The Unified Modeling Language*, series Lecture Notes in Computer Science, R. France and B. Rumpe, Eds., volume 1723, Berlin, Heidelberg: Springer-Verlag, 1999, pages 430–444, ISBN: 9783540667124. DOI: 10.1007/3-540-46852-8_31 (cited on page 25).

[176] A. Lindsay, D. Downs, and K. Lunn, "Business processes - attempts to find a definition", *Information and Software Technology*, volume 45, number 15, pages 1015–1019, 2003, ISSN: 0950-5849. DOI: 10.1016/S0950-5849(03)00129-0 (cited on page 15).

[177] S. Ling and H. Schmidt, "Time petri nets for workflow modelling and analysis", in *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, volume 4, 2000, 3039–3044vol.4. DOI: 10.1109/ICSMC.2000.884464 (cited on page 121).

[178] G. Lowe, "Breaking and fixing the Needham-Schroeder public-key protocol using FDR", in *Proceedings of the Second International Workshop on Tools and Algorithms for Construction and Analysis of Systems (TACAS-96*, series Lecture Notes in Computer Science, T. Margaria and B. Steffen, Eds., volume 1055, London, UK, UK: Springer-Verlag, 1996, pages 147–166, ISBN: 978-3-540-61042-7. DOI: 10.1007/3-540-61042-1_43 (cited on page 92).

[179] L. T. Ly, S. Rinderle-Ma, K. Göser, and P. Dadam, "On enabling integrated process compliance with semantic constraints in process management systems", *Information Systems Frontiers*, volume 14, number 2, pages 195–219, Apr. 2012, ISSN: 1387-3326. DOI: 10.1007/s10796-009-9185-9 (cited on page 120).

[180] L. Ly, S. Rinderle-Ma, D. Knuplesch, and P. Dadam, "Monitoring business process compliance using compliance rule graphs", in *On the Move to Meaningful Internet Systems: OTM 2011*, R. Meersman, T. Dillon, P. Herrero, A. Kumar, M. Reichert, L. Qing, B.-C. Ooi, E. Damiani, D. Schmidt, J. White, M. Hauswirth, P. Hitzler, and M. Mohania, Eds., series Lecture Notes in Computer Science, volume 7044, Berlin, Heidelberg: Springer-Verlag, 2011, pages 82–99, ISBN: 978-3-642-25108-5. DOI: 10.1007/978-3-642-25109-2_7 (cited on page 120).

[181] S. L. Mansar and H. A. Reijers, "Best practices in business process redesign: validation of a redesign framework", *Computers in Industry*, volume 56, number 5, pages 457–471, Jun. 2005. DOI: 10.1016/j.compind.2005.01.001 (cited on pages 179, 180).

[182] R. Mardare, C. Priami, P. Quaglia, and O. Vagin, "Model checking biological systems described using ambient calculus", in *Computational Methods in Systems Biology*, series Lecture Notes in Computer Science, V. Danos and V. Schachter, Eds., volume 3082, Berlin, Heidelberg: Springer-Verlag, 2005, pages 85–103. DOI: 10.1007/978-3-540-25974-9_8 (cited on page 91).

[183] R. J. Mayer, P. D, C. Michael, K. Painter, and P. S. Dewitte, *IDEF Family of Methods for Concurrent Engineering and Business Re-engineering Applications*. New York, USA: Dorset House Publishing, 1992 (cited on page 21).

[184] A. K. A. de Medeiros, "Genetic process mining", PhD thesis, Eindhoven University of Technology, Eindhoven, Netherlands, 2006. [Online]. Available: http://alexandria.tue.nl/extra2/200611953.pdf (cited on pages 183, 187, 236, 241).

[185] J. Mendling, G. Neumann, and M. Nüttgens, "A Comparison of XML Interchange Formats for Business Process Modeling", in *EMISA 2004*, F. Feltz, Ed., volume 56, Bonn: Ges. für Informatik, 2004, pages 129–140+ (cited on page 18).

[186] J. Mendling, H. A. Reijers, and W. M. P. van der Aalst, "Seven process modeling guidelines (7pmg)", *Information and Software Technology*, volume 52, number 2, pages 127–136, Feb. 2010, ISSN: 0950-5849. DOI: 10.1016/j.infsof.2009.08.004 (cited on pages 36, 37, 40, 63).

[187] J. Mendling, "Transactions on petri nets and other models of concurrency ii", in, K. Jensen and W. M. Aalst, Eds., Berlin, Heidelberg: Springer-Verlag, 2009, ch. Empirical Studies in Process Model Verification, pages 208–224, ISBN: 978-3-642-00898-6. DOI: 10.1007/978-3-642-00899-3_12 (cited on page 36).

[188] M. V. Meulen, *Definitions for Hardware and Software Safety Engineers*. Secaucus, NJ, USA: Springer-Verlag, 2000, ISBN: 1852331755 (cited on page 156).

[189] P. Milgrom and J. Roberts, "Complementarities and fit strategy, structure, and organizational change in manufacturing", *Journal of Accounting and Economics*, volume 19, number 23, pages 179–208, 1995, ISSN: 0165-4101. DOI: 10.1016/0165-4101(94)00382-F (cited on page 14).

[190] S. P. Miller, M. W. Whalen, and D. D. Cofer, "Software model checking takes off", *Communications of the ACM*, volume 53, number 2, pages 58–64, Feb. 2010, ISSN: 0001-0782. DOI: 10.1145/1646353.1646372 (cited on page 97).

[191] R. Milner, *A Calculus of Communicating Systems*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1982, ISBN: 0387102353 (cited on pages 40, 41, 294).

[192] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1998, ISBN: 0262631857 (cited on page 198).

[193] M. Z. Muehlen and J. Recker, "How much language is enough? theoretical and practical use of the business process modeling notation", in *Proceedings of the 20th international conference on Advanced Information Systems Engineering*, series Conference on Advanced Information Systems Engineering 2008, Berlin, Heidelberg: Springer-Verlag, 2008, pages 465–479, ISBN: 978-3-540-69533-2. DOI: 10.1007/978-3-540-69534-9_35 (cited on pages 27, 32, 36, 38, 48–50, 58, 95).

[194] M. Muehlen and D. Ho, "Service process innovation: a case study of BPmn in practice", in *Hawaii International Conference on System Sciences, Proceedings of the 41st Annual*, 2008, pages 372–372. DOI: 10.1109/HICSS.2008.388 (cited on page 38).

[195] S. Mukherjee and A. Chakraborty, "Automated fault tree generation: bridging reliability with text mining", in *Reliability and Maintainability Symposium, 2007. RAMS '07. Annual*, IEEE, Jan. 2007, pages 83–88. DOI: 10.1109/RAMS.2007.328096 (cited on page 240).

[196] N. A. Mulyar, W. M. P. van der Aalst, A. H. M. ter Hofstede, and N. C. Russell, "Towards a wpsl: a critical analysis of the 20 classical workflow control-flow patterns", BPM Center, External Report BPM-06-18, 2006. [Online]. Available: http://is.tm.tue.nl/staff/wvdaalst/BPMcenter/reports/2006/BPM-06-18.pdf (cited on page 30).

[197] R. M. Needham and M. D. Schroeder, "Using encryption for authentication in large networks of computers", *Communications of the ACM*, volume 21, number 12, pages 993–999, Dec. 1978, ISSN: 0001-0782. DOI: 10.1145/359657.359659 (cited on page 92).

[198] M. Netjes, W. M. P. van der Aalst, and H. A. Reijers, "Analysis of resource-constrained processes with colored petri nets", in *Sixth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, K. Jensen, Ed., series DAIMI, volume 576, 2005, pages 251–266. [Online]. Available: http://www.daimi.au.dk/CPnets/workshop05/cpn/papers/NetjesAalstReijersfinal.pdf (cited on page 121).

[199] M. Netjes, H. A. Reijers, and W. M. van der Aalst, "Supporting the BPM lifecycle with FileNet", in *EMMSAD Workshop at the 18th International Conference on Advanced Information Systems Engineering (CAiSE'06)*, T. Latour and M. Petit, Eds., 2000, pages 497–508 (cited on pages 93, 295).

[200] O. Nicolae, M. Cosulschi, A. Giurca, and G. Wagner, "Towards a BPMN semantics using UML models", in *Business Process Management Workshops*, series LNBIP, D. Ardagna, M. Mecella, J. Yang, W. Aalst, J. Mylopoulos, M. Rosemann, M. J. Shaw, and C. Szyperski, Eds., volume 17, Berlin, Heidelberg: Springer-Verlag, 2009, pages 585–596, ISBN: 978-3-642-00328-8. DOI: 10.1007/978-3-642-00328-8_59 (cited on pages 29, 33, 49).

[201] C. R. Nielsen, E. H. Andersen, and H. R. Nielson, "Static validation of a voting protocol", *Electronic Notes in Theoretical Computer Science*, volume 135, number 1, pages 115–134, 2005, Proceedings of the Second Workshop on Automated Reasoning for Security Protocol Analysis (ARSPA 2005) Automated Reasoning for Security Protocol Analysis 2005, ISSN: 1571-0661. DOI: 10.1016/j.entcs.2005.06.001 (cited on page 86).

[202] F. Nielson, H. R. Nielson, and C. Hankin, *Principles of Program Analysis*. Secaucus, NJ, USA: Springer-Verlag, 1999, ISBN: 3540654100 (cited on page 86).

[203] F. Nielson, H. R. Nielson, C. Priami, and D. Rosa, "Static analysis for systems biology", in *Proceedings of the Winter International Synposium on Information and Communication Technologies*, series WISICT '04,

Cancun, Mexico: Trinity College Dublin, 2004, pages 1–6. [Online]. Available: http://dl.acm.org/citation.cfm?id=984720.984768 (cited on page 86).

[204] F. Nielson and H. Nielson, "Model checking is static analysis of modal logic", in *Foundations of Software Science and Computational Structures*, series Lecture Notes in Computer Science Denmark, L. Ong, Ed., volume 6014, Berlin, Heidelberg: Springer-Verlag, 2010, pages 191–205, ISBN: 978-3-642-12031-2. DOI: 10.1007/978-3-642-12032-9_14 (cited on pages 95, 228, 238).

[205] H. R. Nielson and F. Nielson, "A monotone framework for CCS", *Computer Languages, Systems and Structures*, volume 35, number 4, pages 365–394, 2009, ISSN: 1477-8424. DOI: 10.1016/j.cl.2008.07.001 (cited on page 86).

[206] T. Nipkow, M. Wenzel, and L. C. Paulson, *Isabelle-HOL: a proof assistant for higher-order logic*. Berlin, Heidelberg: Springer-Verlag, 2002, ISBN: 3-540-43376-7 (cited on page 82).

[207] Object Management Group, "Business process model and notation (BPMN) 2.0", Object Management Group, Needham MA, USA, Standards Document formal/2011-01-03, Jan. 2011. [Online]. Available: http://www.omg.org/spec/BPMN/2.0/ (cited on pages 7, 9, 26, 27, 37, 38, 44, 48, 51, 53–57, 59, 62, 63, 70, 72, 154, 294).

[208] Object Management Group, "OMG Unified Modeling Language (OMG-UML), infrastructure", Object Management Group, Needham MA, USA, Standards Document formal/2011-08-05, Aug. 2011. [Online]. Available: http://www.omg.org/spec/UML/2.4.1/ (cited on pages 25, 26, 44, 299).

[209] Object Management Group, "UML profile for BPMN 2 processes", Object Management Group, Needham MA, USA, Standards Document dtc/2013-04-01, Jun. 2013. [Online]. Available: http://www.omg.org/spec/BPMNProfile/1.0/ (cited on pages 29, 33).

[210] E. R. Olderog and M. Schenke, "Design of real-time systems: interface between duration calculus and program specifications", in *Structures in Concurrency Theory*, series Workshops in Computing, J. Desel, Ed., Springer London, 1995, pages 32–54, ISBN: 978-3-540-19982-3. DOI: 10.1007/978-1-4471-3078-9_3 (cited on page 161).

[211] Organization for the Advancement of Structured Information Standards (OASIS), "Web services business process execution language (WS-BPEL) version 2.0", Organization for the Advancement of Structured Information Standards (OASIS), Standards Document WSBPEL-v2.0-OS, Apr. 2007. [Online]. Available: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html (cited on pages 19, 27, 59, 118, 293, 299).

[212] S. Oswald and R. Caldwell, "Dispensing error rate after implementation of an automated pharmacy carousel system", *American Journal Of Health-system Pharmacy*, volume 64, number 13, pages 1427–1431, Jul. 2007, ISSN: 1079-2082. DOI: 10.2146/ajhp060313 (cited on page 2).

[213] M. A. Ould, *Business Processes: Modelling and Analysis for Re-engineering and Improvement*. New Jersey, USA: John Wiley & Sons, 1995, ISBN: 978-0-471-95352-4 (cited on page 22).

[214] C. Ouyang, M. Dumas, and A. H. M. T. Hofstede, "Pattern-based translation of BPMN process models to BPEL web services", *International Journal of Web Services Research*, volume 5, number 1, pages 42–62, 2007. DOI: 10.1.1.143.3118 (cited on pages 118, 125, 130).

[215] C. Ouyang, M. Dumas, A. H. M. ter Hofstede, and W. M. P. van der Aalst, "From BPMN process models to BPEL web services", in *Proceedings of IEEE International Conference on Web Services*, Washington, DC, USA: IEEE Computer Society, 2006, pages 285–292, ISBN: 0-7695-2669-1. DOI: 10.1109/ICWS.2006.67 (cited on pages 27, 28, 33, 49, 53, 118, 125, 130).

[216] A. Papoulis and U. S. Pillai, *Probability, Random Variables and Stochastic Processes*. McGraw-Hill Science/Engineering/Math, Dec. 14, 2001, ISBN: 0072817259 (cited on page 46).

[217] D. Parker. (Dec. 2013). Prism website, [Online]. Available: http://www.prismmodelchecker.org/ (cited on pages 33, 105, 202).

[218] D. A. Parker, "Implementation of symbolic model checking for probabilistic systems", PhD thesis, University of Birmingham, Birmingham, United Kingdom, 2003. [Online]. Available: http://etheses.bham.ac.uk/229/ (cited on page 101).

[219] M. Pešić, D. Bošnački, and W. M. P. van der Aalst, "Enacting declarative languages using LTLs: avoiding errors and improving performance", in *Proc. SPIN 2010*, series SPIN'10, Berlin, Heidelberg: Springer-Verlag, 2010, pages 146–161, ISBN: 3-642-16163-4, 978-3-642-16163-6. DOI: 10.1007/978-3-642-16164-3_11 (cited on pages 181, 199).

[220] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1981, ISBN: 0136619835 (cited on pages 23, 29, 121, 297).

[221] K. Phalp and M. Shepperd, "Quantitative analysis of static models of processes", *Journal of Systems and Software*, volume 52, number 2-3, pages 105–112, Jun. 2000, ISSN: 0164-1212. DOI: 10.1016/S0164-1212(99)00136-3 (cited on page 22).

[222] B. C. Pierce, *Types and Programming Languages*. Cambridge, MA, USA: MIT Press, 2002, ISBN: 0-262-16209-1 (cited on page 86).

[223] H. Pilegaard, "Language based techniques for systems biology", PhD thesis, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby, 2007. [Online]. Available: http://www2.imm.dtu.dk/pubdb/p.php?5352 (cited on page 86).

[224] A. Platzer and J.-D. Quesel, "European Train Control System: a case study in formal verification", in *11th International Conference on Formal Engineering Methods: Formal Methods and Software Engineering, ICFEM 2009*, K. Breitman and A. Cavalcanti, Eds., series LNCS, volume 5885, Springer, 2009, pages 246–265. DOI: 10.1007/978-3-642-10373-5_13 (cited on page 95).

[225] A. Platzer and J.-D. Quesel, "Keymaera: a hybrid theorem prover for hybrid systems (system description)", in *Automated Reasoning*, series Lecture Notes in Computer Science, A. Armando, P. Baumgartner, and G. Dowek, Eds., volume 5195, Springer Berlin Heidelberg, 2008, pages 171–178, ISBN: 978-3-540-71069-1. DOI: 10.1007/978-3-540-71070-7_15 (cited on page 95).

[226] A. Pnueli, "The temporal semantics of concurrent programs", in *Semantics of Concurrent Computation*, series Lecture Notes in Computer Science, G. Kahn, Ed., volume 70, Springer Berlin Heidelberg, 1979, pages 1–20, ISBN: 978-3-540-09511-8. DOI: 10.1007/BFb0022460 (cited on page 89).

[227] M. E. Porter, *Competitive Advantage: Creating and Sustaining Superior Performance*. New York, USA: Simon and Schuster, 2008, ISBN: 9781416595847 (cited on page 14).

[228] M. E. Porter, "What is strategy?", *Harvard Business Review*, volume 74, number 6, pages 61–78, Nov. 1996, ISSN: 00178012. DOI: 10.1098/rspb.2008.0355 (cited on pages 14, 293).

[229] M. E. Porter and V. E. Millar, "How information gives you competitive advantage", *Harvard Business Review*, volume 63, number 4, pages 149–162, 1985, ISSN: 00178012. DOI: 10.1038/bdj.2007.481 (cited on page 14).

[230] D. Prandi, P. Quaglia, and N. Zannone, "Formal analysis of BPMN via a translation into COWS", in *Proc. of the 10th international conf. on Coordination models and languages*, series COORDINATION 2008, Berlin, Heidelberg: Springer-Verlag, 2008, pages 249–263, ISBN: 3-540-68264-3, 978-3-540-68264-6. DOI: 10.1007/978-3-540-68265-3_16 (cited on pages 29, 33, 49, 120, 122).

[231] S. Project. (Dec. 2013). Avalonedit website, [Online]. Available: https://github.com/icsharpcode/SharpDevelop/wiki/AvalonEdit (cited on page 204).

[232] R. Pugliese and F. Tiezzi, "A calculus for orchestration of web services", *Journal of Applied Logic*, volume 10, number 1, pages 2–31, 2012, ISSN: 1570-8683. DOI: 10.1016/j.jal.2011.11.002 (cited on page 122).

[233] F. Puhlmann and M. Weske, "Investigations on soundness regarding lazy activities", in *Business Process Management*, series Lecture Notes in Computer Science, S. Dustdar, J. Fiadeiro, and A. Sheth, Eds., volume 4102, Berlin, Heidelberg: Springer, 2006, pages 145–160, ISBN: 978-3-540-38901-9. DOI: 10.1007/11841760_11 (cited on page 119).

[234] J. Puustjärvi and L. Puustjärvi, "Automating the coordination of electronic prescription processes", in *2006 8th International Conference on e-Health Networking, Applications and Services (Healthcom 2006)*, Aug. 2006, pages 147–151, ISBN: 9780780397040. DOI: 10.1109/HEALTH.2006.246436 (cited on page 48).

[235] J. Puustjärvi and L. Puustjärvi, "Automating the dissemination of information entities to healthcare professionals", in *Advances in Information Technology*, B. Papasratorn, W. Chutimaskul, K. Porkaew, and V. Vanijja, Eds., series Communications in Computer and Information Science, volume 55, Berlin, Heidelberg: Springer-Verlag, 2009, pages 123–132, ISBN: 978-3-642-10392-6. DOI: 10.1007/978-3-642-10392-6_12 (cited on page 48).

[236] J. Queille and J. Sifakis, "Specification and verification of concurrent systems in CESAR", in *International Symposium on Programming*, series Lecture Notes in Computer Science, M. Dezani-Ciancaglini and U. Montanari, Eds., volume 137, Berlin, Heidelberg: Springer-Verlag, 1982, pages 337–351. DOI: 10.1007/3-540-11494-7_22 (cited on page 89).

[237] A. A. Rad, M. Benyoucef, C. E. Kuziemsky, and A. A. Rad, "An evaluation framework for business process modeling languages in healthcare", *Journal of Theoretical and Applied Electronic Commerce Research*, volume 4, number 2, pages 1–19, 2 Aug. 2009, ISSN: 0718-1876. DOI: 10.4067/S0718-18762009000200002 (cited on pages 3, 38, 48, 224).

[238] J. C. Recker, "Opportunities and constraints: the current struggle with BPMN", *Business Process Management Journal*, volume 16, number 1, pages 181–201, 2010. DOI: 10.1108/14637151011018001 (cited on pages 27, 49, 95).

[239] J. C. Recker, M. Rosemann, M. Indulska, and P. Green, "Business process modeling : a comparative analysis", *Journal of The Association for Information Systems*, volume 10, number 4, pages 333–363, Apr. 2009. [Online]. Available: http://eprints.qut.edu.au/20105/ (cited on page 38).

[240] J. Recker, M. Indulska, M. Rosemann, and P. Green, "Do process modelling techniques get better? a comparative ontological analysis of BPmn", in *Proceedings 16th Australasian Conference on Information Systems*, B. Campbell, J. Underwood, and D. Bunker, Eds., Sydney, Australia: Australasian Chapter of the Association for Information Systems, 2005. [Online]. Available: http://eprints.qut.edu.au/2879/ (cited on page 27).

[241] J. Recker, M. Indulska, M. Rosemann, and P. F. Green, "How good is BPMN really? insights from theory and practice.", in *ECIS*, J. Ljungberg and M. Andersson, Eds., Association for Information Systems, 2006, pages 1582–1593. [Online]. Available: http://is2.lse.ac.uk/asp/aspecis/20060136.pdf (cited on page 38).

[242] J. Recker and J. Mendling, "Lost in business process model translations: how a structured approach helps to identify conceptual mismatch", in *Research Issues in Systems Analysis and Design, Databases and Software Development*, K. Siau, Ed., Hershey, Pennsylvania: IGI Publishing, 2007, pages 227–259, ISBN: 978-1-59904-927-4. DOI: 10.4018/978-1-59904-927-4.ch009 (cited on page 27).

[243] G. Reggio and R. Wieringa, "Thirty one problems in the semantics of UML 1.3 dynamics", in *OOPSLA 99: Workshop on Rigorous Modeling and Analysis with the UML: Challenges and Limitations*, R. France, J. Bruel, B. Henderson-Sellers, A. Moreira, and B. Rumpe, Eds., Nov. 1999. [Online]. Available: http://doc.utwente.nl/61817/ (cited on page 25).

[244] H. A. Reijers, "Design and control of workflow processes: business process management for the service industry", PhD thesis, Technical University of Eindhoven, Eindhoven, The Netherlands, 2003. [Online]. Available: http://dl.acm.org/citation.cfm?id=1767746 (cited on page 122).

[245] A. Riazanov and A. Voronkov, "The design and implementation of VAMPIRE", *AI Communications*, volume 15, pages 91–110, 2 Aug. 2002, ISSN: 0921-7126 (cited on pages 80, 82).

[246] D. Riehle and H. Züllighoven, "Understanding and using patterns in software development", *Theory and Practice of Object Systems*, volume 2, number 1, pages 3–13, 1996, ISSN: 1096-9942. DOI: 10.1002/(SICI)1096-9942(1996)2:1<3::AID-TAPO1>3.0.CO;2-\# (cited on page 29).

[247] A. Rogge-Solti and M. Weske, "Prediction of remaining service execution time using stochastic petri nets with arbitrary firing delays", in *Service-Oriented Computing - 11th International Conference, ICSOC 2013, Berlin, Germany, December 2-5, 2013, Proceedings*, S. Basu, C.

Pautasso, L. Zhang, and X. Fu, Eds., series Lecture Notes in Computer Science, volume 8274, Springer, 2013, pages 389–403, ISBN: 978-3-642-45004-4. DOI: 10.1007/978-3-642-45005-1_27 (cited on page 121).

[248] M. G. Rojo, E. Rolon, L. Calahorra, F. O. Garcia, R. P. Sanchez, F. Ruiz, N. Ballester, M. Armenteros, T. Rodriguez, and R. M. Espartero, "Implementation of the business process modelling notation (BPMN) in the modelling of anatomic pathology processes", in *Proceedings of the 9th European Congress on Telepathology and 3rd International Congress on Virtual Microscopy*, volume 3 (Suppl 1), London, UK: BioMed Central Ltd, Jul. 2008. DOI: 10.1186/1746-1596-3-S1-S22 (cited on pages 48, 224).

[249] E. Rolón, F. García, F. Ruíz, M. Piattini, and L. Calahorra, "Healthcare process development with BPMN", in *Handbook of Research on Developments in E-Health and Telemedicine: Technological and Social Perspectives*, S. R. Cruz-Cunha M. M. Tavares A. J., Ed., Talca, Chile: Facultad de Ingeniería, Universidad de Talca, 2010, pages 1024–1047, ISBN: 978-3-642-19588-4. DOI: 10.4018/978-1-61520-670-4.ch049 (cited on pages 3, 38, 48, 224).

[250] J. W. Ross, P. Weill, and D. C. Robertson, *Enterprise architecture as strategy: Creating a foundation for business execution*. Cambridge, MA: Harvard Business School Press, 2006, ISBN: 1-591-39839-8 (cited on page 15).

[251] S. Roy, A. Sajeev, S. Bihary, and A. Ranjan, "An empirical study of error patterns in industrial business process models", *IEEE Transactions on Services Computing*, volume 99, number PrePrint, 2013, ISSN: 1939-1374. DOI: 10.1109/TSC.2013.10 (cited on page 159).

[252] N. Russell, A. H. ter Hofstede, and W. M. van der Aalst, "NewYAWL: specifying a workflow reference language using coloured petri nets", in *Eighth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools - CPN'07*, K. Jensen, Ed., The contents of this conference can be freely accessed online via the conference's web page (see hypertext link)., Aarhus, Denmark: Department of Computer Science, University of Aarhus, Denmark, 2007. [Online]. Available: http://eprints.qut.edu.au/15557/ (cited on pages 30, 31, 33, 223).

[253] N. Russell, W. M. P. van der Aalst, A. H. M. ter Hofstede, and P. Wohed, "On the suitability of UML 2.0 Activity diagrams for business process modelling", in *Proceedings of the 3rd Asia-Pacific Conference on Conceptual Modelling - Volume 53*, series Asia-Pacific Conference on Conceptual Modelling (APCCM 2006), Hobart, Australia: Australian Computer Society, Inc., 2006, pages 95–104, ISBN: 1-920-68235-X. [Online]. Available: http://eprints.qut.edu.au/25752/ (cited on page 33).

[254] N. Russell and A. H. M. Hofstede, "NewYAWL: towards workflow 2.0", in *Transactions on Petri Nets and Other Models of Concurrency II*, series Lecture Notes in Computer Science, K. Jensen and W. Aalst, Eds., volume 5460, Springer Berlin Heidelberg, 2009, pages 79–97, ISBN: 978-3-642-00898-6. DOI: 10.1007/978-3-642-00899-3_5 (cited on page 30).

[255] N. Russell, A. H. M. T. Hofstede, and N. Mulyar, "Workflow controlflow patterns: a revised view", BPM Center, External Report BPM-06-22, 2006. [Online]. Available: http://www.workflowpatterns.com/documentation/documents/BPM-06-22.pdf (cited on page 30).

[256] T. C. Ruys, "Optimal scheduling using branch and bound with spin 4.0", English, in *Model Checking Software*, series Lecture Notes in Computer Science, T. Ball and S. K. Rajamani, Eds., volume 2648, Springer Berlin Heidelberg, 2003, pages 1–17, ISBN: 978-3-540-40117-9. DOI: 10.1007/3-540-44829-2_1 (cited on page 143).

[257] A. Schäfer, "Combining real-time model-checking and fault tree analysis", in *Proceedings of the 2003 International Symposium of Formal Methods Europe, FME 2003*, K. Araki, S. Gnesi, and D. Mandrioli, Eds., series Lecture Notes in Computer Science, volume 2805, Berlin, Heidelberg: Springer-Verlag, 2003, pages 522–541, ISBN: 978-3-540-40828-4. DOI: 10.1007/978-3-540-45236-2_29 (cited on page 161).

[258] A.-W. Scheer and M. Nüttgens, "ARIS architecture and reference models for business process management", in *Business Process Management, Models, Techniques, and Empirical Studies*, W. M. P. van der Aalst, J. Desel, and A. Oberweis, Eds., series Lecture Notes in Computer Science, volume 1806, Springer, 2000, pages 376–389, ISBN: 3-540-67454-3. DOI: 10.1007/3-540-45594-9_24 (cited on pages 93, 293, 295).

[259] D. A. Schmidt, "Data flow analysis is model checking of abstract interpretations", in *Proceedings of the 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, series POPL '98, San Diego, California, USA: ACM, 1998, pages 38–48, ISBN: 0-89791-979-3. DOI: 10.1145/268946.268950 (cited on pages 95, 228, 238).

[260] K. Schmidt, "LoLA: a low level analyser", in *Proceedings of the 21st International Conference on Application and Theory of Petri Nets*, series ICATPN'00, Berlin, Heidelberg: Springer-Verlag, 2000, pages 465–474, ISBN: 3-540-67693-7. [Online]. Available: http://dl.acm.org/citation.cfm?id=1754589.1754619 (cited on page 120).

[261] F. B. Schneider, "Byzantine generals in action: implementing fail-stop processors", *ACM Transactions on Computer Systems*, volume 2, number 2, pages 145–154, May 1984, ISSN: 0734-2071. DOI: 10.1145/190.357399 (cited on page 162).

[262] H. Shimazaki and S. Shinomoto, "A method for selecting the bin size of a time histogram", 6, volume 19, Cambridge, MA, USA: MIT Press, Jun. 2007, pages 1503–1527. DOI: 10.1162/neco.2007.19.6.1503 (cited on pages 46, 126, 129).

[263] S. Shingo, *A study of the Toyota production system from an industrial engineering viewpoint*, Revised Edition. Portland, OR: Productivity Press, 1989, Originally published as Study of Toyota production system (1981), ISBN: 0915299178 (cited on pages 3, 17, 240).

[264] A. J. Simons and I. Graham, "30 Things that go wrong in object modelling with UML 1.3", in *Behavioral Specifications of Businesses and Systems*, series The Springer International Series in Engineering and Computer Science, H. Kilov, B. Rumpe, and I. Simmonds, Eds., volume 523, New York, USA: Springer-Verlag, 1999, pages 237–257, ISBN: 978-1-4613-7383-4. DOI: 10.1007/978-1-4615-5229-1_17 (cited on page 25).

[265] N. Skrypnyuk, F. Nielson, and H. Pilegaard, "Static analysis of IMC", *Journal of Logic and Algebraic Programming*, volume 81, number 4, pages 522–540, 2012, ISSN: 1567-8326. DOI: 10.1016/j.jlap.2012.03.011 (cited on pages 86, 296).

[266] A. Smith, *An Inquiry into the Nature and Causes of the Wealth of Nations*. Oxford, UK: Oxford University Press, 1776, Oxford World's Classics Edition (*published 2008*), ISBN: 978-0-19-953592-7 (cited on page 15).

[267] J. M. Spivey, *The Z notation: a reference manual*. Hertfordshire, UK, UK: Prentice Hall International (UK) Ltd., 1992, ISBN: 0-13-978529-9 (cited on pages 28, 119).

[268] R. A. Stephans, "Fault tree analysis", in *System Safety for the 21st Century*, R. A. Stephans, Ed., New Jersey, USA: John Wiley & Sons, Inc., 2005, pages 169–188, ISBN: 9780471662549. DOI: 10.1002/0471662542.ch15 (cited on pages 155, 157, 158).

[269] G. Sutcliffe, "The TPTP problem library and associated infrastructure: the FOF and CNF parts, v3.5.0", *Journal of Automated Reasoning*, volume 43, number 4, pages 337–362, Dec. 2009, ISSN: 0168-7433. DOI: 10.1007/s10817-009-9143-8 (cited on pages 82, 299).

[270] F. W. Taylor, *The Principles of Scientific Management*, series History of Economic Thought Books. McMaster University Archive for the History of Economic Thought, 1911, ISBN: 1596058897 (cited on page 44).

[271] A. Thums and G. Schellhorn, "Model checking FTA", in *Proceedings of the 2003 International Symposium of Formal Methods Europe, FME 2003*, series Lecture Notes in Computer Science, K. Araki, S. Gnesi, and D. Mandrioli, Eds., volume 2805, Berlin, Heidelberg: Springer-Verlag, 2003, pages 739–757, ISBN: 978-3-540-40828-4. DOI: 10.1007/978-3-540-45236-2_40 (cited on page 161).

[272] K. Tumay, "Business process simulation", in *Proceedings of the 27th conference on Winter simulation*, series WSC '95, Washington, DC, USA: IEEE Computer Society, 1995, pages 55–60, ISBN: 0-7803-3018-8. DOI: 10.1145/224401.224421 (cited on pages 93, 298).

[273] P. Turney and T. C. I. of Management Accountants, *Activity Based Costing: The Performance Breakthrough*, series Finance and Taxation Series. Kogan Page, 1996, ISBN: 9780749418816 (cited on page 18).

[274] US Department of Defense, "Military standard: defitions of terms for reliability and maintainability", MIL 721C, 1981 (cited on page 156).

[275] A. Valmari, "A stubborn attack on state explosion", *Formal Methods in System Design*, volume 1, number 4, pages 297–322, 1992, ISSN: 0925-9856. DOI: 10.1007/BF00709154 (cited on page 98).

[276] W. M. P. Van Der Aalst, A. H. M. T. Hofstede, and M. Weske, "Business process management: a survey", in *Proceedings of the 1st International Conference on Business Process Management*, W. van der Aalst *et al.*, Eds., series LNCS, volume 2678, Berlin, Heidelberg: Springer-Verlag, 2003, pages 1–12, ISBN: 3-540-40318-3. DOI: 1761141.1761143 (cited on pages 18, 237).

[277] M. Vardi, "Automatic verification of probabilistic concurrent finite state programs", in *Foundations of Computer Science, 1985., 26th Annual Symposium on*, 1985, pages 327–338. DOI: 10.1109/SFCS.1985.12 (cited on page 248).

[278] K. Vergidis, A. Tiwari, and B. Majeed, "Business process analysis and optimization: beyond reengineering", *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, volume 38, number 1, pages 69–82, Jan. 2008. DOI: 10.1109/TSMCC.2007.905812 (cited on pages 18, 22, 44, 180).

[279] B. Victor and F. Moller, "The mobility workbench - a tool for the pi-calculus", in *Proceedings of the 6th International Conference on Computer Aided Verification*, series Lecture Notes in Computer Science, D. L. Dill, Ed., volume 818, London, UK, UK: Springer-Verlag, 1994, pages 428–440, ISBN: 978-3-540-58179-6. DOI: 10.1007/3-540-58179-0_73 (cited on page 119).

[280] H. Völzer, "A new semantics for the inclusive converging gateway in safe processes", in *Proceedings of the 8th International Conference on Business Process Management*, R. Hull, J. Mendling, and S. Tai, Eds., series Lecture Notes in Computer Science, volume 6336, Berlin, Heidelberg: Springer-Verlag, 2010, pages 294–309, ISBN: 978-3-642-15617-5. DOI: 10.1007/978-3-642-15618-2_21 (cited on page 27).

[281] M. Westergaard, D. Fahland, and C. Stahl, "Grade/CPn: a tool and temporal logic for testing colored petri net models in teaching", in *Transactions on Petri Nets and Other Models of Concurrency VIII*, series Lecture Notes in Computer Science, M. Koutny, W. Aalst, and A. Yakovlev, Eds., volume 8100, 2013, pages 180–202, ISBN: 978-3-642-40464-1. DOI: 10.1007/978-3-642-40465-8_10 (cited on page 33).

[282] D. J. White, *Markov decision processes*. New Jersey, USA: John Wiley & Sons, 1993, ISBN: 9780471936275 (cited on pages 43, 44, 106, 114).

[283] W. J. White, A. C. O'Connor, and B. R. Rowe, "Economic impact of inadequate infrastructure for supply chain integration", NIST, Gaithersburg MD, USA, Planning Report 04-2, May 2004. [Online]. Available: http://www.nist.gov/director/planning/upload/report04-2.pdf (cited on pages 17, 45).

[284] A. Wiles, "Modular elliptic curves and Fermat's last theorem", *The Annals of Mathematics*, volume 141, number 3, pages 443–551, May 1995, ISSN: 0003486X. DOI: 10.2307/2118559 (cited on page 81).

[285] S. Williams, "Business process modeling improves administrative control", *Automation*, pages 44–50, Dec. 1967 (cited on page 21).

[286] K. Wissing, "Static analysis of dynamic properties - automatic program verification to prove the absence of dynamic runtime errors.", in *GI Jahrestagung (2)*, R. Koschke, O. Herzog, K.-H. Rödiger, and M. Ronthaler, Eds., series LNI, volume 110, GI, 2007, pages 275–279, ISBN: 978-3-88579-204-8. [Online]. Available: http://subs.emis.de/LNI/Proceedings/Proceedings110/gi-proc-110-048.pdf (cited on page 88).

[287] P. Wohed, W. Aalst, M. Dumas, A. Hofstede, and N. Russell, "On the suitability of BPMN for business process modelling", in *Business Process Management*, series Lecture Notes in Computer Science, S. Dustdar, J. L. Fiadeiro, and A. P. Sheth, Eds., volume 4102, Berlin, Heidelberg: Springer-Verlag, 2006, pages 161–176, ISBN: 978-3-540-38901-9. DOI: 10.1007/11841760_12 (cited on page 27).

[288] J. Womack, D. Jones, and D. Roos, *The Machine That Changed the World: The Story of Lean Production - Toyota's Secret Weapon in the Global Car Wars That Is Now Revolutionizing World Industry*. New York, USA: Simon and Schuster, 2007, ISBN: 9781416554523 (cited on pages 17, 18).

[289] P. Y. H. Wong and J. Gibbons, "A process semantics for BPMN", in *Proc. of the 10th International Conf. on Formal Methods and Software Engineering*, series International Conference on Formal Methods and Software Engineering 2008, Berlin, Heidelberg: Springer-Verlag, 2008, pages 355–374, ISBN: 978-3-540-88193-3. DOI: 10.1007/978-3-540-88194-0_22 (cited on pages 27, 28, 49, 53, 112, 119).

[290] P. Y. H. Wong and J. Gibbons, "A relative timed semantics for BPMN", *Electronic Notes in Theoretical Computer Science*, volume 229, number 2, pages 59–75, 2009, Proceedings of the 7th International Workshop on the Foundations of Coordination Languages and Software Architectures (FOCLASA 2008), ISSN: 1571-0661. DOI: 10.1016/j.entcs.2009.06.029 (cited on page 121).

[291] Workflow Management Coalition, "XML process definition language", Workflow Management Coalition, Hingham, MA USA, Standards Document WFMC-TC-1025, Oct. 2008. [Online]. Available: http://www.wfmc.org/xpdl.html (cited on pages 26, 299).

[292] M. T. K. Wynn, "Semantics, verification, and implementation of workflows with cancellation regions and OR-joins", PhD thesis, Queensland University of Technology, Brisbane, Australia, 2006. [Online]. Available: http://yawlfoundation.org/documents/MoeWynn_Thesis_FinalVersion.pdf (cited on page 64).

[293] K. Xu, Y. Liu, and C. Wu, "BPSL modeler – visual notation language for intuitive business property reasoning", *Electronic Notes in Theoretical Computer Science*, volume 211, pages 211–220, Apr. 2008, ISSN: 1571-0661. DOI: 10.1016/j.entcs.2008.04.043 (cited on page 239).

[294] YAWL Foundation. (Dec. 2013). CPN model of YAWL's semantics, [Online]. Available: http://www.yawlfoundation.org/sites/default/files/newYAWL-rev2.cpn (cited on pages 30, 33, 223).

[295] J.-H. Ye, S.-X. Sun, W. Song, and L.-J. Wen, "Formal semantics of BPMN process models using YAWL", in *Proceedings of the 2008 Second International Symposium on Intelligent Information Technology Application - Volume 02*, Washington, DC, USA: Institute of Electrical and Electronics Engineers, Dec. 2008, pages 70–74, ISBN: 978-0-7695-3497-8. DOI: 10.1109/IITA.2008.68 (cited on pages 27, 29, 49).

[296] C. Zhou and M. R. Hansen, *Duration Calculus: A Formal Approach to Real-Time Systems*, series Monographs in Theoretical Computer Science (An EATCS Series). Berlin, Heidelberg: Springer-Verlag, 2004, ISBN: 3540408231 (cited on pages 161, 295).

[297] C. Zott, R. Amit, and L. Massa, "The business model: theoretical roots, recent developments, and future research", IESE Business School, IESE Research Papers D/862, 2010. [Online]. Available: http://EconPapers.repec.org/RePEc:ebg:iesewp:d-0862 (cited on pages 14, 15).

Glossary

Key technical terms and names used in this thesis:

Application Programming Interface specifies how some software components should interact with each other.

Architecture of Integrated Information Systems is an approach to enterprise modelling. It offers methods for analysing processes by means of simulation, and takes a holistic view of process design, management, work flow, and application processing (Main Source: [258])

Büchi Automata are a type of ω-automaton, which extends a finite automaton to infinite inputs. In essence, Büchi automata recognize infinite word versions of regular languages.

Binary Decision Diagram are a data structure that is used to represent a Boolean function (Main Source: [39])

Business process is a series of related activities aimed at executing the components of a business model in a measurable manner (Main Source: [228])

Business Process Diagram is based on a flowcharting technique tailored for creating graphical models of business process operations. It is a notation that is readily understandable by all business users, from the business analysts that create the initial drafts of the processes, to the technical developers responsible for implementing the technology that will perform those processes, and finally, to the business people who will manage and monitor those processes. (See Definition 3.9)

Business Process Execution Language short for Web Services Business Process Execution Language (WS-BPEL) is an OASIS standard executable language for specifying actions within business processes with web services (Main Source: [211])

Business Process Management is the activity of supporting business processes using methods, techniques and software to design, enact, control and analyse operational processes involving humans, organizations, applications, documents and other sources of information (Main Source: [159])

Business Process Model and Notation is a specific business process modelling language developed by the OMG group (Main Source: [207])

Business Process Re-Engineering is a business management strategy, originally pioneered in the early 1990s, focusing on the analysis and design of workflows and processes within an organization. BPR aimed to help organizations fundamentally rethink how they do their work in order to dramatically improve customer service, cut operational costs, and become world-class competitors (Main Source: [126])

Business Process Re-engineering Life Cycle is a cyclic process of continuous improvement where a TO-BE situation is defined and the gap between the current AS-IS situation and the desired situation is then addressed by modifying the processes which need to be changed in order to achieve the desired situation. (Main Source: [123])

C++ is a well-established statically typed, free-form, multi-paradigm, compiled, general-purpose programming language.

Calculus of Communicating Systems is a process calculus introduced by Robin Milner [191]. Its actions model indivisible communications between exactly two participants. The formal language includes primitives for describing parallel composition, choice between actions and scope restriction. CCS is useful for evaluating the qualitative correctness of properties of a system such as deadlock or livelock.(Main Source: [191])

Communicating Sequential Processes is a formal language for describing patterns of interaction in concurrent systems. Programs in the original CSP were written as a parallel composition of a fixed number of sequential processes communicating with each other strictly through synchronous message-passing. However, considerable development has since happened and many extensions has been developed.(Main Source: [137])

Computation Tree Logic is a branching-time temporal logic in which the model of time is a tree-like structure in which the future is not determined; there are different paths in the future, any one of which might be an actual path that is realised.(Main Source: [39])

Continuous Stochastic Logic is temporal logic for reasoning about Continuous-time Markov chains.

Continuous-Time Markov Chain is a branching-time temporal logic in which the model of time is a tree-like structure in which the future is not determined; there are different paths in the future, any one of which might be an actual path that is realised.(Main Source: [39])

Counter Example Guided Abstraction Refinement is a counter example guided abstraction refinement that attempts to prove the properties on a system by first simplifying it. The approach begins checking with a coarse (imprecise) abstraction of an model and iteratively refines it.(Main Source: [76])

Dafny is an imperative object-based language with built-in specification constructs designed to support the static verification of programs. It is imperative, sequential, supports generic classes, dynamic allocation, and inductive datatypes. The Dafny verifier is run as part of the compiler. As such, a programmer interacts with it much in the same way as with the static type checker when the tool produces errors, the programmer responds by changing the program's type declarations, specifications, and statements.(Main Source: [9])

Discrete-time Markov Chain is a mathematical system that undergoes transitions from one state to another, between a finite or countable number of possible states. (See Definition A.1)

Duration Calculus is an interval logic for reasoning about real-time systems.(Main Source: [296])

Event-driven Process Chain is a type of flowchart used for business process modelling. Event-driven Process Chains can be used for configuring an enterprise resource planning (ERP) implementation,[1] and for business process improvement. The Event-driven Process Chain method was developed within the framework of Architecture of Integrated Information Systems (ARIS) by August-Wilhelm Scheer at the Institut für Wirtschaftsinformatik at the Universität des Saarlandes in the early 1990s. - Main Source: [258])

eXtensible Markup Language is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.

Failures-Divergences Refinement is a refinement checking software tool, designed to check formal models expressed in the Communicating sequential processes (CSP) process algebra. FDR is developed at the University of Oxford.(Main Source: [60])

Fault Tree Analysis is a top down, deductive reasoning failure analysis in which an undesired state of a system is analysed using boolean logic to combine a series of lower-level fault events leading to a larger system failure(Main Source: [97])

FileNet is considered to be one of the leading commercial BPM systems. Processes are modelled graphically and tasks are assigned to work queues. It is aimed at simulation of business processes(Main Source: [199])

GNU General Public License is a widely used free software license, which guarantees end users (individuals, organizations, companies) the freedoms to use, study, share (copy), and modify the software. Software that ensures that these rights are retained is called free software. The license was originally written by Richard Stallman of the Free Software Foundation (FSF) for the GNU project.

Information Technology is the application of computers and telecommunications equipment to store, retrieve, transmit and manipulate data, often in the context of a business or other enterprise. The term is commonly used as a synonym for computers and computer networks, but it also encompasses other information distribution technologies such as television and telephones.

Interactive Markov Chains is Markov chains in which transition probabilities are functions of population frequencies in the states. (Main Source: [265])

Intravenous bag is the infusion of liquid substances directly into a vein. It is used to correct electrolyte imbalances, to deliver medications, for blood transfusion or as fluid replacement. Intravenous therapy can also be used for chemotherapy.

JAVA is a general-purpose, concurrent, class-based, object-oriented computer programming language that is specifically designed to have as few implementation dependencies as possible.

Labelled Transition System is an abstract machine used in the study of computation. The machine consists of a labelled with labels chosen from a set; the same label may appear on more than one transition. If the label set is a singleton, the system is essentially unlabelled, and a simpler definition that omits the labels is possible.(Main Source: [39])

Labelled Transition System is an adjacency matrix style representation of a *BPD* employed as a genotype representation in the optimisation of business processes. (See Section 8.2.5)

Linear Temporal Logic is a modal temporal logic with modalities referring to time. In LTL, one can encode formulae about the future of paths and is sometimes called propositional temporal logic.(Main Source: [39])

Markov Decision Process is a mathematical framework for modelling decision making in situations where outcomes are partly random and partly under the control of a decision maker (actor). It is a discrete time stochastic control process. (See Appendix A.5)

Markov Reward Model Checker is a probabilistic model checking tool. It supports reward extensions of PCTL and CSL (PRCTL and CSRL), and allows for the automated verification of properties concerning long-run

and instantaneous rewards as well as cumulative rewards. MRMC has been developed by the Formal Methods and Tools (FMT) group at the University of Twente, The Netherlands and the Software Modelling and Verification (MOVES) group at RWTH Aachen University, Germany. (Main Source: [151])

Modelling and Analysis of Real Time and Embedded systems is an extension of the UML language.

MOdest TOol EnviRonment is a probabilistic model checking tool for the MoDeST modelling language developed at Saarland University. (Main Source: [50])

Multiple Terminal Binary Decision Diagram is a refinements on the Binary Decision Diagram (BDD) data structure giving way to a number of related graphs. (Main Source: [39])

NEXPTIME is the set of decision problems that can be solved by a non-deterministic Turing machine using time $O(2^{p(n)})$ for some polynomial $p(n)$, and unlimited space.

NuSMV is a reimplementation and extension of SMV symbolic model checker, the first model symbolic checking tool based on Binary Decision Diagrams.(Main Source: [74])

Object Management Group is the consortium responsible for CORBA (Common Object Request Broker Architecture), Unified Modeling Language (UML), and Model-Driven Architecture (MDA).

OPPAAL-SMC is a probabilistic model checking tool. The name is derived from the first three letters of Uppsala University (UPP) and Aalborg University (ALL), while SMC stands for the Statistical Model Checking extension of this tool.(Main Source: [63])

Ordered Binary Decision Diagram is a type of Binary Decision Diagram that ensures the variables appear in the same order along all paths from the root to the leaves. (Main Source: [62])

Performance Evaluation of Parallel Programs is a model checker which can provide numerical evaluation of processes' performance. (Main Source: [59])

Performance Evaluation Process Algebra is a stochastic process algebra designed for modelling computer and communication systems introduced by Jane Hillston.(Main Source: [135])

Petri-net is a directed bipartite graph, in which the nodes represent transitions and places. Directed arcs describe which places are pre- and/or post-conditions for which transitions occurs. (Main Source: [220])

Probabilistic Computation Tree Logic is an extension of computation tree logic (CTL) which allows for probabilistic quantification of described properties. (See Appendix A.2 - Main Source: [131])

Probabilistic Symbolic Model Checker is a formal verification software tool for the modelling and analysis of systems that exhibit probabilistic behaviour. PRISM is developed at the university of Oxford.(Main Source: [170])

Probabilistic Timed Automata are an extension of Markov decision processes with clocks and constraints on clocks. It is an extension of timed automata with (discrete) probabilistic choice.

Process Graph Fundamentally, models of business processes focus on the sequencing of tasks, where a task is an abstraction of a piece of work which is to be performed. Process graphs encapsulate this notion and are formally defined in the thesis. They are employed as an overarching definition that captures this sequencing of tasks within a well-defined mathematical structure.(See Definition 3.1)

PROTOS is a modelling and analysis tool developed by Pallas Athena and it is mainly applied for the specification of their in-house business processes. Processes can be analysed with respect to data, user and control logic perspectives with mean, 90% and 99% confidence intervals of utilization rates, waiting times, service times, throughput times and costs. PROTOS is a project by Oulu University Secure Programming Group (OUSPG).(Main Source: [272])

Rewards Data values associated with states of a system which are accumulated as different states or transitions in the system are traversed. Both types of rewards are formally defined in this thesis.(See Section 3.2.3)

SPIN is a general tool for verifying the correctness of distributed software models in a rigorous and mostly automated fashion. It was written by Gerard J. Holzmann and others in the original Unix group of the Computing Sciences Research Center at Bell Labs, beginning in 1980. Systems to be verified are described in Promela and properties to be verified are expressed as Linear Temporal Logic (LTL).(Main Source: [140])

Statespace is a representation of a system model where each state of the statespace corresponds to a unique configuration of the system and transitions between states describe the evolution of the system's configurations.(Main Source: [39])

Stochastic BPMN Optimisation Analysis Tool is a software service which allows for the analysis and automated radical optimisation of business processes. This is the name of the software tool created by the author.(See Chapter 9)

Stochastic Hybrid Automata At the core of the Modest Toolset is the model of networks of stochastic hybrid automata (SHA), which combine non-deterministic choices, continuous system dynamics, stochastic decisions and timing, and real-time behaviour, including non-deterministic delays.

The Modelling and Description Language for Stochastic and Timed Systems is a specification formalism for describing stochastic real-time systems. (Main Source: [50])

The Technical University of Denmark is a university just north of Copenhagen, Denmark. It was founded in 1829 at the initiative of Hans Christian Ørsted as Denmark's first polytechnic, and is today ranked among Europe's leading engineering institutions, and the best engineering university in Scandinavia.

Thousands of Problems for Theorem Provers is a library of test problems for automated theorem proving (ATP) systems. (Main Source: [269])

UML-Statecharts are an enhanced realization of the mathematical concept of a finite automaton in computer science applications as expressed in the Unified Modelling Language notation.(Main Source: [208])

Unified Modeling Language Systems Modeling Language is a general-purpose modeling language for systems engineering applications. It supports the specification, analysis, design, verification and validation of a broad range of systems and systems-of-systems. SysML is defined as an extension of a subset of the Unified Modeling Language (UML) using UML's profile mechanism

Unified Modelling Language is a standardized general-purpose modelling language, developed by the Object Management Group.(Main Source: [208])

UPPAAL is an integrated tool environment for modelling, validation and verification of real-time systems modelled as networks of timed automata, extended with data types (bounded integers, arrays, etc.). Developed in collaboration between the Department of Information Technology at Uppsala University (UPP) in Sweden and the Department of Computer Science at Aalborg University (AAL) in Denmark.(Main Source: [43])

Web Services Business Process Execution Language is a standard for formally describing business processes and business interaction protocols. WS-BPEL was designed to extend the Web Services interaction model to support business transactions.(Main Source: [211])

Weighted Metric Temporal Logic is a temporal logic for reasoning about weighted timed automata and is an extensions of CTL that allows for the incorporation of relative weights of separate CTL formula.

XML Process Definition Language is a format standardized by the Workflow Management Coalition (WfMC) to interchange business process definitions between different workflow products, i.e. between different modeling tools and management suites. XPDL defines an XML schema for specifying the declarative part of workflow / business process.(Main Source: [291])

Yet Another Workflow Language is a workflow language based on the Workflow patterns. The language is supported by a software system that includes an execution engine, a graphical editor and a worklist handler. The system is available as Open source software under the lesser GPL license. (See Section 2.3.2.3 - Main Source: [23])